

University of Nottingham
UK | CHINA | MALAYSIA

A Lightweight Cascaded CNN Architecture for Efficient Image Classification in Edge Platforms

ISURU BANDARA SAMARAKOON

Student ID: 20071451

Supervised by

Prof. Haider Abbas Almurib
Prof. Anandan Shanmugam

Thesis submitted to the University of Nottingham for the degree of
Doctor of Philosophy

February 2026

Declaration

I, Isuru Bandara Samarakoon, hereby declare that the work presented in this thesis entitled " **A Lightweight Cascaded CNN Architecture for Efficient Image Classification in Edge Platforms**" is the result of my own research and it has not been submitted elsewhere for any other degree at this University or any other institution. To the best of my knowledge, the thesis contains no materials from previously published work other than those cited and acknowledged.

Signature _____

Date _____ / _____ / _____

Abstract

Convolutional neural networks (CNNs) have demonstrated remarkable success in image classification, driving their widespread adoption in research and industry. While deeper networks introduce through more complex computations to achieve higher accuracy, they also incur longer inference times. This creates a fundamental trade-off between accuracy and inference latency, as increased model complexity slows down individual classifications and, consequently, the overall performance of vision-based systems. This thesis presents a novel cascaded CNN architecture designed to achieve an optimal balance between classification accuracy, inference speed, and computational efficiency for real-time operation on edge devices. The proposed two stage design integrates a lightweight main CNN for rapid initial classification and a secondary CNN that is selectively invoked to refine predictions from the main network. The architecture incorporates several optimization strategies, including efficient layer design, layer fusion, pruning, and hardware aware design, enabling the model to operate effectively within the limited memory and processing capabilities of embedded systems. The model was developed and trained using the MATLAB Deep Learning Toolbox, with performance evaluated across 200 object classes relevant to autonomous guidance and service robotics in the intended application of an airport environment.

The proposed CNN model was trained on a dataset derived from the Ima-

geNet ILSVRC-2012 dataset, using 70% of the data for training and 30% for validation. The test dataset comprised randomly selected images from the internet, as well as real-time images captured by a camera connected to the computer. The performance of the proposed cascaded neural network model was evaluated based on classification accuracy and execution time. Experimental assessments were conducted on both a GPU enabled workstation and a Raspberry Pi 4B to evaluate real-world performance. For a wide range of test data presented, the proposed network architecture enhanced the inference speed by 6.43% compared to the AlexNet while maintaining a 95% Top-1 classification accuracy for tests conducted on 200 object classes and proved to be within the expected performance criteria. Additional benchmarking across multiple state-of-the-art CNN models shows that the proposed system offers superior efficiency in terms of accuracy and inference, underscoring its suitability for real-time robotic guidance and other embedded vision applications. Furthermore, the proposed network model was tested on standard computer systems and Raspberry Pi models, demonstrating deployability and compatibility on edge devices where the test data suggest that the proposed fast executing network model is best suited for edge devices with a minimum computational resources environment.

Acknowledgements

I am forever grateful to my supervisors, Prof. Haider Abbas and Asst. Prof. Anandan Shanmugam for their immense support and guidance throughout this study. Their experience, knowledge and patience blended with motivation and encouragement provided me to successfully achieve my goals during the study period. Their rich knowledge and wealth of experience always blessed me to direct during hardest times to find answers and to make a better approach. I practiced in a true research environment where I learned how to effectively conduct, appreciate, and excel in research. Amid the pandemic, challenging environment however made it achievable because of the unwavering guidance and motivation of my supervisors. I would like to thank my family members for their continued support during my study period encouraging me during the difficult times, bearing happiness and sadness over rises and falls with me. I am forever grateful to my family members and appreciate anyone who has supported me directly or indirectly. I would also like to thank all the staff members in the Faculty of Engineering who supported me during the research study. I would also like thank to colleagues at the university for their support from the beginning of this journey.

Contents

Abstract **i**

Acknowledgements **iv**

Chapter 1 Introduction **2**

1.1 Background 3

1.2 Problem Statement 5

1.3 Hypothesis 6

1.4 Proposed solution 6

1.5 Aim and Objectives 9

1.6 Research Scope 10

1.7 Summary of contribution 11

1.8 Thesis Overview 13

Chapter 2 Literature Review **15**

2.1 Introduction to the chapter 17

2.2 General introduction of Convolutional Neural Networks . . . 18

2.3 Study of other network models 23

2.4 Shifts in CNN development 27

2.5 Factors affecting the execution speed of a CNN model 30

2.6 Trade-offs between speed and memory in batch processing . 33

2.7 CNN architectural optimizations 35

2.8 Different options to enhance the inference speed. 40

2.9 Scientific approaches to enhance the processing speed of CNNs. 43

2.10 Lightweight CNN architectures 47

2.11 Energy efficient CNN architectures 49

2.12 Benchmarking and evaluation matrices of CNNs 51
2.13 Cascaded CNN architectures 53
2.14 Advantages, Challenges and Applications of Cascaded CNN architectures 55
2.15 Chapter Summary 60
Chapter 3 Methodology 62
3.1 Introduction to the Chapter 64
3.2 Motivation to the novel approach 65
3.3 AlexNet overview 67
3.4 Proposed Cascaded CNN architecture 70
3.5 Main network architecture 74
3.6 Secondary network architecture 76
3.7 Major CNN layers and operational principles 77
3.8 Methodology 106
3.9 Implementation 118
3.10 Proof of the concept 134
3.11 Chapter Summary 150
Chapter 4 RESULTS AND DISCUSSION 153
4.1 Introduction to the Chapter 154
4.2 Results of the cascaded network model 155
4.3 Performance analysis of sub network 161
4.4 Combined network's performance 165
4.5 Classification accuracies 170
4.6 CNN Model Benchmarking 181
4.7 Chapter Summary 186
Chapter 5 CONCLUSIONS AND FUTURE WORK 188
5.1 Main contributions 188
5.2 Model effectiveness 192

5.3 Research challenges and dataset limitations 193
5.4 Summary of chapters 195
5.5 Future work 196
Bibliography 198
Appendices **227**
Chapter A Appendices **228**
A.1 Performance Evaluation of a Proposed Two Stage Cascaded Convolutional Neural Network Architecture 228

List of Tables

2.1 CNN architecture with convolution, pooling, and classification layers 21

2.2 Comparison of CNN Models: Parameters and Operations . . 46

3.1 Learnable parameters of Alexnet 65

3.2 Comparison of Network Models: Parameters and Execution Time 66

3.3 Main network model specifications 75

3.4 Specifications of secondary network model 77

3.5 Learnable parameters in convolution layers of secondary network 113

3.6 Computations of secondary network 113

3.7 Learnable parameters of main network 114

3.8 Computations in main network model 114

3.9 Hardware specifications of the computer 130

3.10 computations involved in convolutional layers of Alex net. . 136

3.11 computations involved in convolutional layers of main network 138

3.12 computations involved in convolutional layers of sub network 139

3.13 computations from ReLU layers of AlexNet 140

3.14 computations from ReLU layers of Main network 141

3.15 computations from ReLU layers of sub network 141

3.16 computations from Maxpool layers of Alexnet 142

3.17 computations from Maxpool layers of Main network 143

3.18 computations from Maxpool layers of sub network 143
3.19 computations from Fully connected layers of Alex net 145
3.20 Fully Connected Layer Computations 145
3.21 computations from Fully connected layers of main network . 145
3.22 comparison of computations 146
3.23 cascaded network execution time for different classes 148
3.24 transfer learned AlexNet execution time for different classes. 149

4.1 Learnable Parameters of Main Network 156
4.2 Summary of Measurable Quantities 156
4.3 Execution Time for Different Layers 157
4.4 Comparison of theoretical and practical execution time for main network 157
4.5 Classes combination and parameters for Alexnet and cascaded network 161
4.6 Computations and learnable parameters in sub network . . . 162
4.7 Secondary network parameters 162
4.8 Comparison of main and secondary network 163
4.9 Main network and sub network classes combination 166
4.10 Learnable parameter selection for main and sub network classes 167
4.11 Model Evaluation Summary of CNN Architectures using real-time data 182

List of Figures

2.1 CNN architecture of LeNet-5 network model [1] 19

2.2 Non-normalized data used in LeNet-5 from MNIST dataset [1] 19

2.3 AlexNet network architecture [2] 20

2.4 Inception module [3] 31

3.1 Comparison of Learnable parameters with execution time . . 66

3.2 Flowchart of the proposed cascaded CNN architecture 73

3.3 Main network architecture 74

3.4 Secondary convolutional neural network architecture 76

3.5 Convolution operation on input image with a kernel producing output feature map [4] 79

3.6 Output feature map generation from input image with weighted kernel [5] 80

3.7 Input image to the network(left). 81

3.8 Output of the image input layer (Right) 81

3.9 color cube input image(left), corresponding output from the image input layer(right) 82

3.10 Output feature map of 1st convolution layer 82

3.11 Feature map of 2nd convolution layer 83

3.12 Sigmoid activation response 87

3.13 ELU activation responses for $\alpha = 1$ 89

3.14 ELU activation responses for variable hyperparameter values 89

3.15 ReLU activation function response 90

3.16 a). standard neural network b). after application of dropout [6] 96
3.17 output feature map a) Without dropout. b). with dropout p=0.5 (right) [6] 96
3.18 inter connection of nodes a). standard network b). dropout network [6] 98
3.19 Deep network designer interface 119
3.20 Deep Network Designer analysis report for the network model 120
3.21 deep network designer training options 121
3.22 Training progress graph 122
3.23 Profiler output for a single execution of the main network and the sub network 129
3.24 Mobile robot platform equipped with a vision unit. (a) The vision unit is mounted on the mobile robot structure. (b) Vision Unit Components: USB Camera, Touch Display Unit, Raspberry Pi 4, and Pi Camera v2.1 133
3.25 Real-time results from dynamic outdoor environment. 133
3.26 comparison of CNN execution times 151

4.1 Main network performance for number of classes up to 100 . 158
4.2 Main network performance for number of classes from 100 to 1000 159
4.3 Sub network performance for the classes from 5 to 100 . . . 164
4.4 Sub network performance for the classes from 100 to 1000 . 164
4.5 Cascaded network performance 168
4.6 Cascaded network performance for 1000 classes 168
4.7 AlexNet network performance for number of classes from 5 to 100 169
4.8 AlexNet network performance for number of classes from 100 to 1000 170

4.9 Cascaded network classification accuracies 176
4.10 Alex net re-trained network classification accuracy 177
4.11 Classification results for real-time images of test objects . . . 179
4.12 Classification results for real-time images of test objects executed on Raspberry Pi 180
4.13 Cascaded network classification results for real-time images of test objects executed on a Raspberry Pi A400 computer. . 180
4.14 Average Confidence by CNN Model 183
4.15 Average Inference Time by CNN Model 184
4.16 Combined comparison of Average Confidence and Inference Time across CNN models, showing dual y-axes with unified legend below the plot. 185

Abbreviations

CNN Convolutional Neural Networks.

CPU Central processing unit.

DNN Deep neural network.

ELU Exponential linear unit.

FPGA Field programmable gate array.

GPU Graphical processing unit.

LRN Local response normalization.

MAC Multiply and accumulate.

MED Mean error distance.

MRED Mean relative error distance.

MSE Mean squared error.

R-CNN Region based convolutional neural network.

ReLU Rectified linear unit.

TPU Tensor processing unit.

Nomenclature

$\hat{x}$	Normalized output
C_{in}	Number of input channels
H_{in}	Input image height
W_{in}	Input image width
α	Hyper parameter for negative inputs
β	Shift parameter of batch normalization
δ	Shift parameter of batch normalization
γ	Scale parameter of batch normalization
σ	Dropout rate
C_{out}	Output channels
e	Base of natural logarithm
H_{out}	Output feature map height
W_{out}	Output feature map width

CHAPTER 1

Introduction

Contents

1.1 Background **3**

1.2 Problem Statement **5**

1.3 Hypothesis **6**

1.4 Proposed solution **6**

1.5 Aim and Objectives **9**

1.6 Research Scope **10**

1.7 Summary of contribution **11**

1.7.1 List of Publications 13

1.8 Thesis Overview **13**

1.1 Background

With the evolution of technology, machines have been extensively used for various domestic and industrial applications. The machines were then developed with the perspective of improved work efficiency and ensured safety to comply or exceed human labor capacity. However, machines were further developed to make their own decisions instead of human controllers. Detection and identification of objects is one of the most challenging tasks for machines without prior knowledge of the application of computers. Linear classifiers, fundamental concept in the field of machine learning used to classify data out of given set of samples. Perceptron, the first linear classifier inspired by the biological neuron was a giant leap in modern deep neural networks and machine learning. With the very basic constituents, input weights and activation function producing a classification has later been inspired by researchers and proposed modifications. LeNet-5 [1] One of the success stories developed by Le Cun et al, a simple convolutional neural network, where the first time of introduction of the Convolution operation to the classification model. Referring to the Lenet-5 architecture, various new models were developed and presented by researchers such as AlexNet [2] and ZFNet [7] with slightly more convolution operation and dense connectivity included compared to Lenet-5 architecture. Researching the advantages and disadvantages of the existing networks researchers designed new architectures such as ResNet [8] and GoogLeNet [3]. Though Lenet-5 was trained on fewer dataset for classification of handwritten digits, AlexNet trained on a large dataset of 1000 classes with large variety in the dataset due to its generalization capacity. Architectural developments such as micro neural network [9] has been inspiration for higher accuracy models such as ResNet-50 and VGG16 which included more convolutional opera-

tions and dense connectivity expecting higher accuracy. When applying these neural network models into the industry [10–14], accuracy is one of the major concerns. However, applications such as automobile pedestrian detections [15] on active safety systems required detection and identification of an object to be completed within minimum time as possible. Thus, research was focused on execution time of the neural network which contributes to the inference of the model. As the convolutional operations are increased, with the intention of obtaining higher accuracy which results increase in complex computations where the Central processing unit (CPU) should elapse more time on computations resulting more inference time or execution time delay which determines how fast the process can produce an output. The number of computations can be reduced in different ways either by model architectural changes or hardware resource modifications. This has been addressed by many different network models such as ZFNet [7]. Improved network architectures can generate accurate results. However, the consistency of the results can be improved by adapting the cascaded architecture [16–24]. Stacking multiple Convolutional Neural Networks (CNN)s in such manner to produce more refined predictions from the prior stages resulting a concise prediction essentially in various applications such as biomedical [18], human detection [21], face recognition [20] etc. importance of convergence of the cascaded CNNs were highlighted in [10] while addressing a practical issue of smaller samples [24]. In this research the main concern is to reduce the inference time by network architectural modifications. It is presented here the importance of reduced number of computations, cascaded network architecture and overall effectiveness of cascaded convolutional neural network.

1.2 Problem Statement

Cascading convolutional neural network model with minimum complex computations involved, produces accurate classification output for a given object with least execution time for the network model. Convolutional neural networks are useful for classification of objects efficiently in machine learning applications. Extensive convolutional operations and dense connections lead to higher accuracies but also results in longer inference time. According to the general trade-off where the higher accuracy leads to more complex models and the faster inference tends to simpler optimized models. Hence, the acceptable balance between accuracy and inference time in CNN models depends on the application domain and the deployment environment. For most industrial and domestic applications, it is expected to produce object classification with adequate accuracy and minimum latency. The proposed model is intended for use in an airport guidance service robot, utilizing an edge device like the Raspberry Pi. It aims to achieve a classification accuracy of approximately 70-85%, with an inference time of less than 200 milliseconds per classification. Since the application is not time-critical, it prioritizes moderate accuracy and low latency while maintaining efficiency on low-power hardware. When addressing the issue of unnecessary inference time offered by the network model, it can be reduced by two approaches as hardware resource modification and architectural modifications. However, to provide a versatile solution for the issue of inference latency, network model architectural modifications are essential. This research work introduces, extensively discuss, evaluates and presents the best solutions for the architectural modifications of cascaded convolutional neural networks.

1.3 Hypothesis

Based on the data presented in sections 3.1 and 2.2, it is clear that the number of layers of a convolutional neural network is increased with the intension of enhancing the classification accuracy, of the model which directly increases the number of complex computations involved and thus the overall execution time of the network also increases. From the literature presented in section 2.3 to achieve higher classification accuracy, a tendency of adapting more computational layers had been a common practice. Interest in feedforward neural networks has shown that classification accuracy can be improved by incorporating more convolutional layers and denser connections. However, this enhancement depends on several factors, including the training dataset, the distribution of weights and biases, stochastic dropouts, and the characteristics of the input data. It is important to note that adding more convolutions leads to increased computations, resulting in longer inference times. This research hypothesizes that cascading multiple lightweight Convolutional Neural Networks (CNNs) with reduced numbers of learnable parameters can significantly minimize overall execution time while maintaining competitive classification accuracy, compared to single deep CNN architectures with higher computational complexity.

1.4 Proposed solution

The primary objective of this research is to design, optimize, and deploy a convolutional neural network (CNN) model that achieves high classification accuracy while ensuring low inference latency on resource-constrained hardware, specifically a Raspberry Pi-based guidance service robot.

The proposed solution is organized into four main stages: designing the model architecture, optimizing the model, preparing and training the dataset, and deploying the model for real-world testing. High-accuracy convolutional neural network (CNN) architectures, such as ResNet, DenseNet, and EfficientNet, achieve impressive classification performance. However, they require significant computational resources, making them impractical for real-time deployment on embedded systems. However, older lightweight models such as SqueezeNet and AlexNet offer faster inference times, but sacrifice some accuracy. A novel architecture is proposed to address existing gaps in the design process. This architecture features a main network and a sub-network model that incorporates elements from modern efficient models, such as depthwise separable convolutions from MobileNet and bottleneck layers from ResNet, to reduce computational complexity. Using fewer parameters and a shallower layer depth to accommodate the processing capabilities and memory limits of the Raspberry Pi. Furthermore, the design balances accuracy and speed through architectural tuning, which includes selecting the appropriate kernel size, reducing feature maps, and optimizing activation functions.

The proposed architecture consists of two cascaded lightweight models, referred to as the main network and the sub network. Both networks are computationally optimized to minimize unnecessary computations, thereby reducing inference time. Pruning the model architecture from both convolutional and dense layers is expected to simplify complex mathematical operations. The main network has a smaller number of learnable parameters, which leads to fewer mathematical computations. Similarly, the sub network also has a comparatively smaller number of parameters, resulting in fewer computations as well. Both network models were individually trained on different datasets, and their performances were evaluated sep-

arately. The cascaded arrangement of the main and sub-networks was tested, and overall performance was measured across various combinations of datasets for both networks. The main network provides broader classifications, while the sub network refines the predictions from the main network, leading to a more precise generalization that is not typically offered by conventional CNN architectures.

The proposed solution employs a dataset with 200 object classifications that represents indoor navigation and service environments. The dataset preparation process includes:

I. Collecting and curating object images relevant to guidance and navigation tasks.

II. Performing image augmentation (rotation, scaling, brightness adjustment) to improve model robustness to environmental variations.

III. Splitting the dataset into training, validation, and test sets to ensure reliable performance evaluation.

The training of the CNN model is conducted on a Windows operating system using a GPU enabled environment within MATLAB software. This setup helps to expedite convergence through hyperparameter tuning, which includes adjusting the learning rate, batch size, and selecting the optimizer.

The final stage involves testing the novel architecture on Windows-based computer systems using both a test dataset and real-time data. Prior to remote deployment, remote debugging will be conducted on the edge device to serve as a preliminary feasibility test. Upon successful testing, the model will be deployed on the target edge device, such as a Raspberry Pi, for compatibility testing. Real-time images will be obtained from a

USB camera connected to both the test computers and the Raspberry Pi units. The performance of the proposed model will be analyzed by measuring inference time, classification accuracy, and system responsiveness under operational conditions. To validate the improvements of the proposed solution, we will compare its performance against the baselines of AlexNet and MobileNet. A successful outcome will demonstrate that the proposed approach achieves the following:

I. Inference times of less than 200 milliseconds per image.

II. An accuracy improvement of at least 6% over AlexNet.

III. Stable and responsive classification, making it suitable for live robotic guidance in indoor environments.

This proposed solution is expected to make significant contributions in several key areas. Firstly, it will introduce a lightweight CNN architecture specifically designed for embedded robotic vision, enabling more efficient and effective visual processing on resource-constrained devices. Secondly, the solution will provide an optimization framework that can be applied to other low-power AI systems, thereby enhancing their performance and scalability. Lastly, it will establish a reproducible evaluation protocol for service robot vision models, ensuring consistency and reliability in assessing and comparing different approaches within this domain.

1.5 Aim and Objectives

This research aims to design and implement a cascaded Convolutional Neural Network (CNN) framework that optimizes the trade-off between classification accuracy and computational efficiency, by reducing model com-

plexity and execution time while maintaining robust performance for real time object recognition in resource constrained computing environments.

The objectives of this research include:

1. To design and develop a lightweight primary Convolutional Neural Network (MainNet) that achieves high classification accuracy while minimizing computational complexity in terms of model parameters, memory requirement, and inference latency.

2. To design and develop a secondary Convolutional Neural Network (SubNet) with enhanced generalization capability for resolving ambiguous classifications while maintaining low computational overhead suitable for real-time deployment.

3. To integrate the MainNet and SubNet into a cascaded CNN architecture that dynamically invokes the subsequent CNN based on the confidence thresholds from the primary stage and evaluate its performance using a domain specific dataset derived from the ImageNet database.

4. To conduct a comprehensive performance analysis of the proposed cascaded model, assessing classification accuracy, confidence, and inference time across multiple object categories, and validating its real-time applicability on edge computing platforms.

1.6 Research Scope

This research focuses on the development and optimization of convolutional neural network (CNN) architectures for real-time image classification on edge devices. The goal is to reduce inference time while maintaining

high classification accuracy. The study includes a thorough review of the evolution of CNNs, from early models like LeNet to modern architectures such as EfficientNet and ConvNeXt. The light weight two stage cascaded network was developed using MATLAB on a standard PC and tested on PC environment, followed by a deployment on edge devices. Experiments were conducted using a curated dataset containing 200 object classes intended for application in the airport environment, focusing on a limited number of objects. Comparisons are made against benchmark models such as AlexNet. This study does not cover object tracking, segmentation, or human-machine interaction. The findings aim to support the development of efficient deep learning models suitable for mobile and embedded systems.

1.7 Summary of contribution

This thesis addresses the challenge of developing a lightweight and fast inference convolutional neural network (CNN) model for real-time object classification applications in resource constrained environments, specifically focusing on guidance service robots powered by edge devices like Raspberry Pi. While deep CNN architectures achieve superior accuracy, their high computational demands often result in unacceptable inference delays on low-power devices. This research aims to minimize inference time while maintaining a classification accuracy above 65%, thereby surpassing the performance of baseline models such as AlexNet. The main contributions of this research are:

1 . Comprehensive Review of CNN Evolution – Conducted an extensive survey tracing the development of CNN architectures from early designs like LeNet to modern high performance models, including Ef-

ficientNet and ConvNeXt. The review emphasizes trends in computational efficiency and the suitability of these models for edge devices.

2 . Design of a Lightweight CNN Model - The proposed CNN architecture is optimized to achieve a 6% improvement in Top-1 accuracy over AlexNet across 200 object classes while significantly reducing inference latency. The pruning strategies have resulted in a 40% reduction in model size without a significant decrease in accuracy, enabling faster, more accurate, and memory efficient execution.

3 . Development of a benchmark dataset for the vision of Service robots – Compiled and curated a dataset that accurately represents indoor navigation and object classification tasks faced by guidance service robots. This dataset facilitates reproducible evaluations of Convolutional Neural Network (CNN) models in this application area. The image data was extracted from the ImageNet ILSVRC dataset.

4 . Performance Evaluation of the Novel Model - The performance evaluation of the novel network model was conducted on both a PC system and an edge device using a test dataset to assess classification accuracy and inference speed. The model was also tested with real-time data from a camera attached to the systems. Furthermore, the optimized CNN model was deployed on the edge device, specifically a Raspberry Pi, conducting real-world trials that demonstrated responsive object classification with inference times under 200 milliseconds. This performance met the operational requirements for live service scenarios.

1.7.1 List of Publications

1 Journal papers Isuru Samarakoon, H. Abbas Almurib and S. Anandan Shanmugam "Performance Evaluation of Two Stage Cascaded Convolutional Neural Network Architecture in Image Classification", Springer Nature Journal of Neural Computing and Applications (NCAA) [Accepted for Publication].

2 Poster presentation Samarakoon, I., Haider, A., Shanmugam, A. (2019) Human Detection and Tracking System for Passenger Guide Mobile Service Robot Platform. Presented in Post Graduate Research showcase, University of Nottingham, Malaysia.

1.8 Thesis Overview

This thesis is divided into five chapters. Chapter 1 introduces the aim of the research, related background, basics of convolutional neural networks, the problem statement, what led for this research, objectives to be achieved to succeed the aim, brief summary of the chapter. Chapter 2 outlines the literature review carried out for convolutional neural networks. Evolution of the convolutional neural networks, introduction to the deep neural networks. What factors affect the accuracy and processing speed, different approaches to speed up the neural network model, impact on network architectural modifications and hardware resource modifications, different approaches to speed up the inference are discussed under this chapter. Chapter 3 describes the methodology of the overall network design. In detail description of main network design, sub network model design, mathematical relation to the number of computations involved. Chapter 4 describes the results

and analysis of the new network model. Individual network's performances, presented. Cascaded model's performances were presented, and the results were compared with one of the most cited convolutional neural network models, AlexNet. Lastly, chapter 5 concludes the thesis by summarizing the key points.

CHAPTER 2

Literature Review

Contents

2.1 Introduction to the chapter **17**

2.2 General introduction of Convolutional Neural Networks **18**

2.3 Study of other network models **23**

2.4 Shifts in CNN development **27**

2.5 Factors affecting the execution speed of a CNN model **30**

2.6 Trade-offs between speed and memory in batch processing **33**

2.7 CNN architectural optimizations **35**

2.8 Different options to enhance the inference speed. . . . **40**

2.9 Scientific approaches to enhance the processing speed of CNNs. **43**

2.10 Lightweight CNN architectures **47**

2.11 Energy efficient CNN architectures **49**

2.12 Benchmarking and evaluation matrices of CNNs . . . **51**

2.13 Cascaded CNN architectures **53**

2.14 Advantages, Challenges and Applications of Cascaded CNN architectures **55**

2.15 Chapter Summary **60**

2.1 Introduction to the chapter

Accuracy and inference speed are prime aspects of object classification applications.Convolutional Neural Networks (CNNs) have emerged as one of the most influential deep learning architectures for solving a wide range of tasks in computer vision. Inspired by the visual cortex of animals with biological perception the perceptron, have evolved from shallow architectures such as LeNet-5 to highly complex deep models such as ResNet and EfficientNet. Learning complex features from raw data has made CNNs essential in modern artificial intelligence.

This chapter reviews the evolution, innovations, and challenges in CNN research. The review begins by introducing the foundational concepts of CNNs with traces the historical development of CNN architectures, highlighting key milestones such as AlexNet, VGG, EfficientNet, ResNet, and recent lightweight and transformer-integrated models. It then discusses various factors affecting the inference speed and various options and attempts to enhance the execution speed. This chapter also includes shifts in CNN development, attention mechanism and presents the lightweight architectures for edge devices. Furthermore, this chapter discusses the architectural optimizations depth wise separable convolutions

The goal of this literature review is to provide a comprehensive understanding of the current landscape of CNN research; limitations associated with them and establish the rationale for the proposed cascaded network in this thesis. The insights drawn from this chapter will guide the development of proposed cascaded convolutional neural network model addressed in the subsequent chapters.

2.2 General introduction of Convolutional Neural Networks

With the industrial revolution it became common practice to develop sophisticated machines to carry out industrial activities in order to ensure productivity efficiency and safety thus replacing manpower. As such systems had to take actions on time with minimum or zero supervision, it was necessary to take essential decisions by the system itself. The detection and identification of different objects in a dynamic environment [25] being an easy task for humankind while the most complicated for a machine [26] [15]. This was inspired by the researchers thus developing classical linear classifiers. Due to the complex computations involved and straight forward results, linear classifiers were limited for practical applications. For certain applications such as mobile robots [25], service robots, medical robots etc. with limited power and resource availability application of classical classifiers was one of great challenge. Perceptron [1] [27] being the simplest artificial neural network developed by Rosenblatt in 1957 inspired by biological neurons formulating the very first linear classifier for supervised tasks. With the rigorous research involved over the time with development of technologies and devices, more flexible classification model based on convolutional neural network, the Lenet-5 [1] was developed by LeCun et al in late 90s. Lenet-5 visualized in Figure 2.1.

LeNet-5 is a simple network architecture consisting of five convolutional layers. Originally designed for detecting handwritten numbers where the network had trained on MNIST dataset [1] inspired by Krizhevsky et al in 2012. Non normalized data example of handwritten data depicted in Figure 2.2. Being a turning point of machine learning, AlexNet [2] had

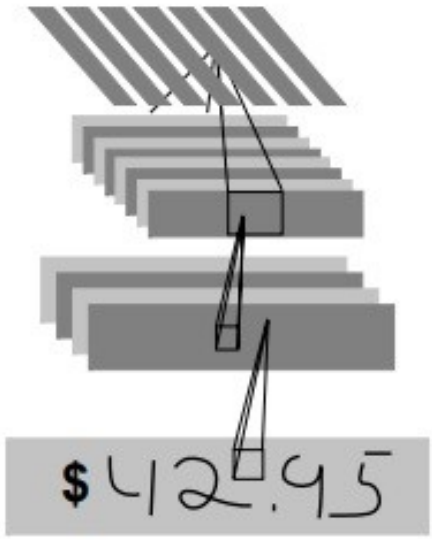


Figure 2.1: CNN architecture of LeNet-5 network model [1]



Figure 2.2: Non-normalized data used in LeNet-5 from MNIST dataset [1]

implemented more dense layers in the network architecture compared to the grand origin of LeNet-5 with only five layers.

Figure 2.3 presents the AlexNet network architecture. The winner of ILSVRC-2012 challenge, the AlexNet has been trained on ImageNet dataset with large number of data and variety in classes has motivated researchers to use more dense networks for better classification which could be seen in models such as ResNet [8] where more sparse connectivity is present.

Although there was a trend to classify more objects with higher accuracy

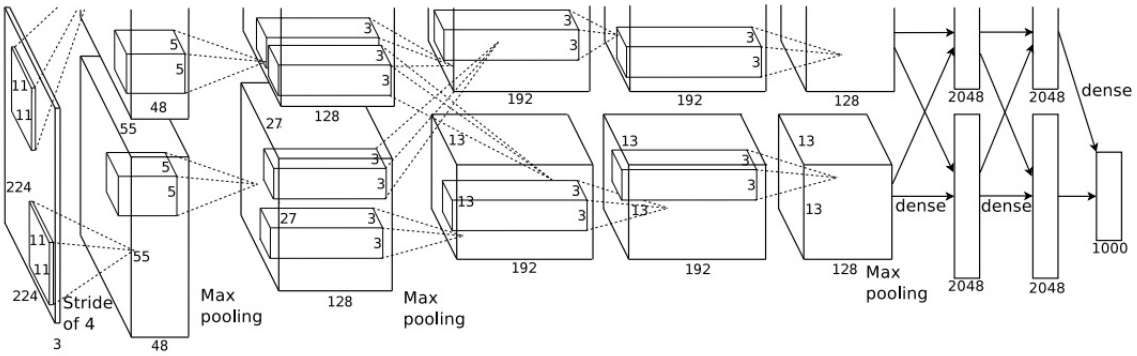


Figure 2.3: AlexNet network architecture [2]

using more complex computations involved, another pragmatic approach was to speed up the process [8] has been adapted in [28] [29]. However, the main intention of reducing the filter size and strides as it was accomplished in [7] was more generalized in [2]. Another novel approach of better classification with faster processing was single shot detectors [28] and reduced filter sizes has been adapted in YOLO [9] by Redmon et al. Filter sizes strides and output sizes of the Darknet-19 which was based on YOLO are presented in Table 2.1.

Table 2.1: CNN architecture with convolution, pooling, and classification layers

Type	**Filters**	**Size/Stride**	**Output**
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2 / 2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2 / 2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2 / 2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2 / 2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2 / 2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1 × 1
Softmax			1000

Impressive results of [28] and [7] had motivated researchers to apply the models in [29] for the automated detection of melanoma. Region proposal network model [30] was adapted in Faster-Region based convolutional neural network (R-CNN) which also proposes faster execution compared to previous network models. Successful detection method of pedestrians [31] has been proposed by Wenbo Lan et al, by adapting the above methods. Also, a deep learning method as the name suggests, more deep or dense layers are present in their network architectures. Despite improving detection accuracy, the processing speed has been enhanced up to 25 frames per second with modified YOLO architecture, producing a new model YOLO-R. Various architectural modifications [32–39] developed by researchers presenting individual generalization capabilities and speed of operation compared with existing models. Novel convolutional technique of fast algorithm [32] which improves response speed and reduced computational complexity. Transfer learning of various CNN models and adaptation for applications [33–35] gives a better overview of performances of various CNN models. J.Y. Ahn and N.I. Cho [36] have presented neural architecture search for multi-scale feature capturing. One of the drawbacks of faster R-CNN architecture, the negative samples [38] highlighted by Y Liu proposing novel algorithm with improved detection accuracy. Comparison of CNN models for classification of communication signals by D. Seo et al where computational complexity and detection performance is compared. Similarly real time people counting applications [40] has been proposed by Peiming et al, of which counting people in real-time dynamic environments. The need of fast execution of network identified by the research community proposing novel approaches such as detection of workers without safety helmet in construction site real-time video achieving a fast processing as fast as 35 frames per second.

2.3 Study of other network models

LeNet-5 It is essential to choose a neural network model that provides high classification accuracy while maintaining minimal execution time, ideally with low latency. LeNet, one of the earliest convolutional neural network architectures, was originally developed for handwriting digit recognition by LeCun et al. LeNet-5 features a straightforward architecture, consisting of two convolutional layers and three fully connected layers, which contributes to its effectiveness in classification tasks.

Mugesh et al. [41], who compared a multi-layer perceptron and a simple CNN with LeNet-5, presented a comparative analysis of various models for handwritten digit recognition. Soni et al. [42] proposed a modified version of the LeNet-5 model for identifying strawberry leaf diseases, achieving a classification accuracy of 95% on the validation dataset. Additionally, Tan et al. [43] enhanced the LeNet-5 architecture by removing its 1x1 convolutional layer, resulting in improved classification accuracy for handwritten digits.

Further improvements were made by Devi et al. [44], who introduced an additional dense layer to the existing LeNet-5 architecture, incorporating batch normalization with ReLU activation and a dropout layer. This modification achieved an impressive accuracy of 98.68% for mushroom classification. While much of the research has focused on improving accuracy, Chu et al. [45] adapted pipeline processing for LeNet-5 to increase processing speed.

It is important to note that the aforementioned studies primarily focus on the classification of a limited number of classes. However, in an airport environment, a system is needed to handle a medium to high number

of classes, ranging from 200 to 300 categories. Consequently, the LeNet architecture was not selected for the network model in this context.

ResNet Among various models, ResNet, developed by He et al. [8], was the first to achieve top performance in the ILSVRC 2015 challenge by introducing residual blocks stacked to create deeper networks for improved classification accuracy. ResNet offers two versions: ResNet-50 and ResNet-101, consisting of 50 and 101 layers, respectively. These deeper residual blocks help minimize error rates and enhance generalization performance, effectively addressing the vanishing gradient problem [46].

Varun et al. [47] modified ResNet for real-time analysis, achieving higher accuracy while tackling data imbalances and improving model interpretability. The transfer learning capabilities of the ResNet architecture were emphasized in [48], where both ResNet-50 and ResNet-101 models were adapted and reached an impressive accuracy of 98.8%. Duan et al. [49] improved the ResNet architecture by integrating multi-parameter and squeeze excitation techniques, resulting in better generalization performance.

However, the presence of deeper residual blocks makes the ResNet architecture computationally expensive [50], leading to higher latency [51]. Extensive computations require sophisticated resources, which makes ResNet unsuitable for low-end mobile computing platforms due to its significant demand for computational power and memory storage [52]. Additionally, ResNet performs best with large datasets, which can lead to overfitting when working with smaller datasets [53]. Given these limitations, we have decided not to pursue ResNet for the proposed architecture.

VGGNet Another CNN architecture VGGNet was developed by Simonyan et al, is known for its use of small 3x3 convolutional kernels. This architecture gained prominence due to its performance in the ImageNet Large

Scale Visual Recognition Challenge in 2014. The simplicity and uniformity of the architecture, characterized by stacked 3x3 convolutions, make it easy to understand and modify [54].

Xu et al [55]. demonstrated the adaptability of the VGG-16 architecture, achieving a high accuracy of 97.7% in target image recognition for underwater search and rescue. Spoorthy et al [56]. compared the generalization capabilities of VGG-16 with other network models, highlighting the benefits of deep architectures for classifying bone fractures in X-ray images, with an accuracy of 95%. Xiangyu et al [57]. emphasized the importance of the transfer learning capabilities of the VGG-16 model, which is particularly useful for modifying existing CNN models.

Despite these advantages, the VGG-16 model has drawbacks, including a large model size [58] and a tendency to overfit on small datasets [59] due to the significant number of learnable parameters.

EfficientNet EfficientNet, developed by Tan et al. [60], is a family of convolutional neural networks (CNNs) known for their accuracy and efficiency, achieved through the use of compound scaling parameters. The advantages of compound scaling have been highlighted by Jagamohan et al. [61], showcasing its effectiveness in image classification tasks.

EfficientNet is particularly suitable for transfer learning; for instance, Gurjot et al. [62] adapted this model for the early detection of pneumothorax, achieving an accuracy rate of 83%. Ziyang et al. [63] successfully utilized EfficientNet for recognizing ischemia and infections in diabetic foot ulcers, outperforming other models such as ResNet, DenseNet, and VGG-16.

The remarkable accuracy of EfficientNet, which reached 98% with a top-1 error rate of 84.4%, is largely attributed to its compound scaling method

that uniformly adjusts the depth, width, and resolution of the neural network. However, it requires a significantly large training dataset.

Another application of EfficientNet in biomedical research was presented by Amreen et al. [64], who focused on classifying leukemia white blood cell images. The performance of EfficientNet-B3, a lightweight model within the EfficientNet family, has been improved with a reduced number of trainable parameters. Preetha et al. [65] adapted a variant of EfficientNet for detecting brain tumors from magnetic resonance images, achieving an F1 score of 0.9934 and an overall accuracy of 99.33%.

Testing all eight variants of EfficientNet, from EfficientNet B0 to EfficientNet B7, revealed that a modified version of EfficientNet B4 offered the best performance on the 2020 Kaggle dataset. Scalability is a key factor in selecting the network model; higher versions are particularly well-suited for larger datasets like ImageNet, enabling them to learn complex patterns, although they are also prone to overfitting and require more computational resources.

Compound scaling effectively adjusts the model architecture based on the input image resolution, which has been a key factor in enhancing the efficiency of the EfficientNet model. However, the EfficientNet architecture also has drawbacks, including a complex design and the computational intensity of its large-scale versions, making them difficult to train from scratch. Additionally, there is a potential for overfitting, especially when working with smaller datasets. While compound scaling is effective for some applications by dynamically changing the network model parameters, it is not always the best solution for every task and dataset. Given these pros and cons, it was decided to proceed with the AlexNet architecture as the foundation.

2.4 Shifts in CNN development

The development of Convolutional Neural Networks (CNNs) has undergone significant changes since their inception. These changes have been driven by advancements in computational resources, the availability of large datasets, and innovative architectural designs. As a result, the performance, scalability, and applicability of CNNs across various domains have improved considerably.

The foundation for these developments can be traced back to the mathematical model of neurons established by Hebbian Learning Rules, proposed by Donald O. Hebb in 1949. This was further reinforced by the introduction of the Perceptron by Frank Rosenblatt in 1958. In 1980, K. Fukushima introduced the Neocognitron, a bio-inspired model that set the stage for the evolution of CNNs, eventually leading to the emergence of dense networks. The Universal Approximation Theorem, proposed by George Cybenko in 1989, highlights the potential of shallow neural networks, particularly when a large number of neurons are present in the layers of a multi-layer perceptron. The foundational work on convolutional neural networks (CNNs) can be traced back to LeNet-5, developed by Yann LeCun et al. in the late 1990s. This model successfully applied convolutional architectures to the recognition of handwritten digits.

Despite its conceptual strength, early CNNs were limited by the available computational resources and the lack of sufficient data and hardware for deeper models. As a result, their adoption was primarily restricted to tasks of low complexity. The vanishing gradient problem, which hinders the training of models and was present in LeNet-5, has been addressed effectively by support vector machines proposed by Corinna Cortes in 1995.

This development temporarily stalled further advancements in CNNs. A significant turning point in deep learning occurred in 2012 with the success of AlexNet, presented by Krizhevsky et al. in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). AlexNet utilized GPU acceleration, the ReLU activation function, and dropout regularization, enabling it to greatly exceed the performance of traditional methods. This achievement marked the resurgence of convolutional neural networks (CNNs) and established them as a leading approach in computer vision research.

Following AlexNet, subsequent models such as GoogLeNet and VGGNet further improved efficiency by strategically deepening the network and enhancing gradient flow. The evolution of convolutional neural networks has shifted from simpler architectures to a greater emphasis on network depth. A major breakthrough in this area was the development of ResNet by He et al., which introduced residual blocks that facilitated the creation of even deeper network models.

With the evolution of the processor technology, as CNNs began to be deployed on mobile and embedded platforms, a shift toward efficiency emerged. Lightweight architectures such as MobileNet, SqueezeNet, and ShuffleNet introduced techniques like depthwise separable convolutions, grouped convolutions, and pruning to reduce computational complexity and memory usage. These models facilitated real-time inference on resource-constrained devices, broadening the practical applicability of CNNs. MobileNet developed by Howard et al [66] highlighting small size and low latency model which can easily be matched to edge devices and embedded vision applications. MobileNet's core layers are concentrated on depth wise separable convolutions. The depthwise separable method factorize a standard convolution into a depthwise convolution and a pointwise convolution. The depthwise separable convolution applies a single kernel per

input channel which captures spatial information independently from each channel. The following pointwise convolution is 1x1 convolution applies across the input channels, which combines the depthwise convolution output form number of discrete channels. Compared to the standard convolution, depthwise separable convolution is beneficial due to lesser learnable parameters which results in less computation. The significant reduction in number of computations has brought the attention of model prioritizing the speed and light weight inference which is best suitable for edge devices to perform real-time application. ShuffleNet developed by Zhang et al. [67] has adapted pointwise group convolution with channel shuffle which significantly reduces computational cost. ShuffleNet prioritize the computational efficiency while maintaining competitive classification accuracy. SqueezeNet developed by Iandola et al. [68] is a deep neural network trained on ImageNet dataset over 1000 classes, known for its small size and classification accuracy with significantly lesser number of learnable parameters compared to the other CNN architectures. The computation efficiency has been achieved by replacing most of 3x3 kernels with 1x1 kernels, reduction of number of input channels to the large kernel layers by decreasing the quantity of parameters. Down sampling late in the network has contributed to maximizing the classification accuracy. The importance of Depthwise separable convolution has been highlighted in [69] where reduction of computation time by 19.1% with a remarkable reduction of on chip memory requirement it has also been implemented on FPGA proving the suitability of edge devices. Ruiheng et al. [70] has proposed novel method based on FPGAs for accelerating depthwise separable convolution network which offers Energy efficiency, reduction of on chip memory requirement and edge device deploy ability. Shaowei et al. [71] has adapted the depthwise separable convolution for classification of petrographic thin section images. Application of DSC has resulted in classification accuracy

99.84%

2.5 Factors affecting the execution speed of a CNN model

As discussed in the previous chapter, the most important operation of any convolutional neural network happens at the sparse connectivity populated by the convolutional units and dense layers where the neurons converge. Inference speeds up methods of the CNN models [25, 26, 72–79] have been extensively discussed in many literatures since the inference time should be minimized for most industrial applications [25, 26]. Mathematically complex computations are being taken at these two stages which contributes significant amount of time consumption by the network model. In general, the factors affecting the processing speed of a CNN can be classified into two broad categories such that,

I. Hardware resource-oriented factors.

II. Model architecture related factors.

The model architecture related factors can be explained as follows.

- Model architecture depth: For a given network if a greater number of sparse connections [80] and dense connections are present such model is known as a deep network. The Deep Convolutional networks introduced by Fukushima [27] where the "neocognitron" consists of cascaded interconnection of layers in the network model. The architecture has been improved in [4] and further simplified in [81]

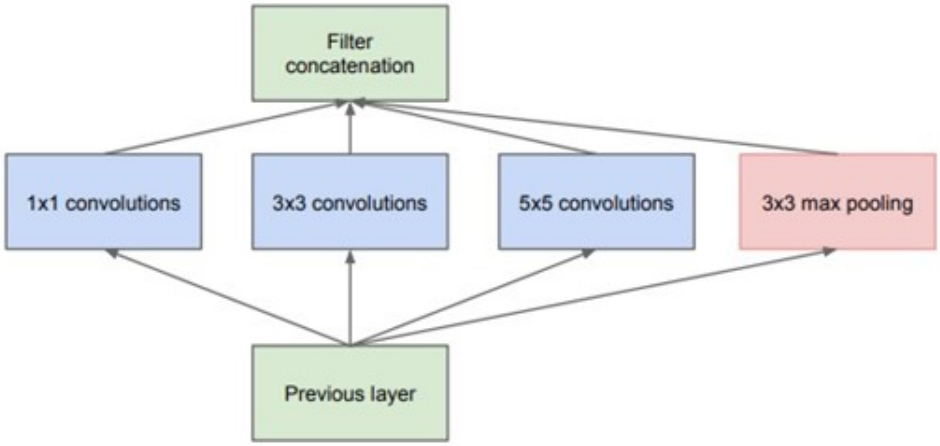


Figure 2.4: Inception module [3]

and [82]. Due to the presence of more convolutional layers and dense layers in a network, more complex computations must be executed, which generally requires more time for the computation [26] and thus longer inference time. Reducing the depth of the network thus leads to fast inference while maintaining fair accuracy.

- Convolutional layer type: Within the convolutional layer, a large number of computations are performed. The kernel size [83] and strides determine the number of complex computations [7] involved in the network. Kernel size is one of the factors that contribute to the number of computations. Thus, the inference depends on the number of convolutional layers present in the network. Importance of kernel size extensively discussed in GoogLeNet [2] architecture where the inception module presented in Figure 2.4 is a remarkable contribution.

- Pooling layers in network architecture: The pooling layers are used after the convolutional layers, essentially after the activations to reduce spatial dimensions before propagating to the next layer thus reducing the computations. Pooling layers must be carefully selected as it leads to loss of information if excessively applied.

- Model weight pruning: reducing unimportant weights results in reduction of computations involved during the network execution and thereby reducing the inference. However, it is also possible to remove less important filters which further reduces model size. As it may lead to the accuracy of the network it has to be carefully conducted.

Hardware resource-oriented factors which affect the speed can be further explained in the following.

- Hardware acceleration devices: Using specialized hardware tools the process of execution can be accelerated. Graphical Processing Units (Graphical processing unit (GPU)) [81] [73] [74] or Tensor Processing Units (Tensor processing unit (TPU)) are the latest accelerators widely used to speed up convolutional neural networks inference. Also, these devices facilitate parallel computation where the workload is distributed among the cores.

- Batch sizing: When processing in parallel it is required to arrange the data in batches [84]. Batch size has better advantage over inference since processing is speed up with more batch size. However, there is a disadvantage with increasing batch size since it requires large memory [85] which is expensive. Hence the is a tradeoff between batch size and computation efficiency.

- Input size and format: Since the image input is the only source of information, alteration of the input results in classification accuracy [36]

as it may lead to distortion of the original data. Reducing the input image size speeds up the execution of the network while resulting in loss of information.

- Caching and memorization: the intermediate results should be stored, which reduces redundant computations. This leads to speeding up the execution process. However, cache memory is an expensive option as the resources should be changed. Therefore, selection of cache must be optimum in terms of performance.

2.6 Trade-offs between speed and memory in batch processing

Batch processing in Convolutional Neural Networks (CNNs) involves dividing the training dataset into smaller groups called batches. These batches are processed one at a time during training, as opposed to using the entire dataset or a single sample at once. This approach strikes a balance between computational efficiency and training stability, making it essential for training deep learning models on a large scale. In deep learning, especially with CNNs, the batch size is a crucial hyperparameter that directly affects various aspects of the training process. GPU memory usage (memory constraint)

I. Training speed (throughput and convergence)

II. Generalization and stability

Memory bandwidth is one of the bottlenecks in GPUs, even when com-

pute units are underutilized. GPUs can be heavy on arithmetic but often starve for memory because the memory subsystem struggles to provide data quickly enough. Larger batch sizes require storing more activations [86], gradients, and immediate feature maps in GPU memory during both forward and backward passes. As a result, memory loading gradually increases with larger batch [87]. Therefore, dynamic batching is implemented based on memory and latency conditions.

Training speed and batch size are factors that depend on hardware. Generally, larger batch sizes lead to higher throughput but also increase latency per sample. This can significantly impact training efficiency. As a rule of thumb, throughput is proportional to batch size. In their study, wieduo et al. [88] demonstrated that increasing batch size enhances training throughput. However, larger batches also contribute to increased latency. Since GPU memory is limited, larger batches require more memory to store activations, feature maps, and gradients. Additionally, using larger batches can increase the risk of poor generalization [89]. A method for selecting the appropriate batch size is proposed in [90], which balances the performance gains associated with larger batches.

The relationship between generalization and stability can be examined concerning batch size, as both factors significantly influence a model's overall performance. Generalization refers to how well a model performs on test data, while stability pertains to whether the training process converges smoothly without divergence or oscillations.

Smaller batch sizes tend to enhance generalization because they introduce gradient noise, which helps the optimizer navigate away from sharp minima [89]. In contrast, larger batch sizes yield more accurate gradient estimates, leading to smoother training that often converges to sharp minima,

which can result in poor generalization on test data. This impact of smaller batches has been highlighted in research by Sunwoo et al [91] which demonstrated improvements in accuracy when using smaller batch sizes.

From a stability perspective, smaller batches may improve generalization but compromise stability [92]. Conversely, larger batches enhance stability while potentially degrading generalization. Therefore, using smaller to medium batch sizes is advisable for edge devices due to memory constraints.

Additionally, the stability of model training can be improved by fine-tuning the learning rate [93]. Studies suggest that using an adaptive learning rate [94] can be particularly beneficial in achieving better stability.

2.7 CNN architectural optimizations

Architectural optimizations refer to techniques aimed at enhancing the speed, accuracy, efficiency, and deployability of convolutional neural network (CNN) models while focusing on specific design constraints, such as those found in edge devices and embedded systems. Various CNN architectural optimization techniques exist, including kernel optimizations, efficient layer design, architectural scaling, attention mechanisms, hybrid designs, and hardware-oriented design. The techniques contributing to lightweight and fast inference model design selected for this research are detailed below.

1. Efficient layer design: The efficient layer design emphasizes optimizations in the kernel convolution of the network model. As discussed in the previous chapter, depth-wise separable convolutions break standard convolutions into two parts: depth-wise and point-wise convolutions. This approach significantly reduces the number of parameters

and computations required. Depth-wise convolutions have been successfully integrated into various network models, such as MobileNet and EfficientNet. Grouped convolution is an efficiency enhancement technique used in network layers. It involves dividing the input channels into groups and convolving each group separately. This approach allows for parallel processing and reduces computational demands while preserving the model's representational capacity.

Saining et al. [95] developed the ResNeXt model, which incorporates grouped convolution to optimize computations and achieved a top-1 error rate of 20.7% across 1,000 classes. Similarly, Mozafari et al. [96] applied single-dimensional grouped convolution to the BERT language model for its fully connected layers, resulting in a 42% reduction in fine-tuning time, demonstrating its effectiveness for edge devices.

Additionally, Tinghuan et al. [97] showed that their method utilizing grouped convolution reduced computational parameters by 64.17% compared to ResNet, while also improving top-1 accuracy by 78.86%.

2. Inverted residual block: Inverted residual blocks are an architectural optimization technique for convolutional neural networks (CNNs), first introduced in MobileNetV2 to create lightweight and efficient models for edge devices. Unlike the residual blocks used in ResNet, an inverted residual block initially expands the input channels, applies depthwise convolution, and then compresses the output, with a residual connection added at the end. The term "inverted" refers to this sequence of operations. Sandler et al [98] proposed the inverted residuals and implemented them in the MobileNetV2 model, achieving memory-efficient inference with good accuracy. Additionally, Yongli et al [99] adapted the inverted residuals to mitigate the

interference from the batch normalization layer in their underwater image enhancement work, resulting in significant accuracy improvements.Biaohua et al [100] demonstrated the effectiveness of inverted residuals in enhancing accuracy for their infrared small target detection model, which highlights their role in reducing interference. Similarly, Ruitao et al [101] adapted inverted residuals for lightweight modulation classification, emphasizing improvements in classification accuracy and reductions in computational costs.

3. Hardware oriented design: Hardware-aware design is a crucial aspect of optimizing convolutional neural network (CNN) architectures. This approach involves creating neural network models that take into account hardware constraints such as memory, computational power, and energy consumption to enhance real-world performance. Standard neural network models that are designed solely for accuracy may not be suitable for resource-constrained environments, as they often have high memory usage, long inference times, and excessive power consumption.

   In hardware-aware design, it is essential to match the compute budget, ensuring that the model operates within the peak floating-point operations per second (FLOPs) or multiply-accumulate (MAC) capabilities of the target device. For instance, a study highlighted in [102] demonstrates that the proposed architecture aligns with the peak FLOPs of a Raspberry Pi device, achieving a mean average precision (mAP) of 96.66% while reducing the model size by 50% through hardware-aware design. Optimization of the memory hierarchy is a crucial aspect of hardware-aware design. It is important to minimize activation and weight storage while maximizing the use of on-chip cache memory, as this significantly impacts energy costs in addition

to memory expenses. Seonwoo et al [103], in their proposed model, highlighted the effects of memory overhead in hardware-aware design, which resulted in a 1.5 times acceleration in inference speed.

Latency and energy-aware neural architecture search (NAS) plays a vital role in hardware-oriented architectural optimizations. This search process is guided by feedback from the hardware. Cai et al [104] proposed a method that involves training a model once and then searching for sub-networks optimized for edge devices.

4. Layer fusion and pruning: Layer fusion and pruning are important techniques for optimizing convolutional neural networks (CNNs). These strategies work together to enhance model performance and facilitate efficient deployment on edge devices. Pruning refers to the process of removing redundant components of the neural network, which helps reduce the model's size and the complexity of computations. This reduction in parameters leads to fewer floating-point operations (FLOPs) and a smaller memory footprint. As a result, lower inference times and reduced memory requirements create ideal conditions for resource-constrained edge devices. Li et al [105] introduced structured filter pruning, demonstrating substantial reductions in FLOPs with minimal loss in accuracy. Their approach achieved a 50% reduction in parameters, resulting in a speed increase of 1.5 times. Radu et al [106] emphasized the importance of performance-aware pruning, showing that naïve channel pruning can paradoxically slow down inference on embedded GPUs unless the pruning patterns align with the hardware kernel implementations. More recently, Hao et al [107] proposed the TECO framework, which extends pruning to multiple dimensions of convolutional neural networks (CNNs), including depth, width, and resolution for embedded hardware. This

method achieved a twofold increase in inference speed with less than a 3% improvement in top-1 accuracy, tested on edge devices such as the AGX Xavier and Jetson Nano. Layer fusion is the process of combining multiple consecutive layers of a neural network into a single computational unit, which helps to avoid intermediate activations and reduces writing to main memory. This approach contributes to decreased memory traffic and, consequently, lower energy consumption. Additionally, it reduces latency while improving throughput, which is particularly important for embedded CPUs and System on Chips (SoCs) that have limited memory bandwidth.

Graph level fusions, such as Conv-BN-ReLU, have been widely adopted in inference compilers, as noted by Xuyi et al [108]. Recent research has highlighted architecture-aware fusion scheduling, as demonstrated by Minyong and Jungwook [109], which shows latency-optimal fusion strategies on embedded accelerators. This has resulted in significant speedups of 24.1% for the ResNet18 model and 53.1% for the MobileNet V2 model.

In the context of the Internet of Things (IoT), Mestre et al [110]. validated the fusion of Conv2D, ReLU, and BN into a unified layer called FConv2D. This combination, when integrated with quantization, achieved a measurable speedup, reducing inference execution time by 1.53 times on resource-constrained ARM Cortex Microcontroller Unit (MCU) devices. Pruning and fusion work in tandem to create an efficient pipeline. Pruning eliminates unnecessary computations, while fusion helps to reduce runtime overhead. When these techniques are combined with quantization, they significantly improve latency, memory usage, and energy efficiency. This combination is particularly advantageous for edge devices, such as the Raspberry Pi.

The CNN architectural optimization techniques discussed above are crucial for real-time object classification tasks on edge devices, as they help achieve a balance between accuracy and efficiency. This research integrates several state-of-the-art CNN optimization methods into the proposed model, resulting in a lightweight and accurate system that is suitable for deployment on resource-constrained edge devices for real-time inference applications.

2.8 Different options to enhance the inference speed.

As we discussed in section 2.5 the factors contribute to the longest execution time, few options can be highlighted to improve the speed of execution of a CNN model such as executing resource changes, minimizing the learnable layers, minimizing kernel sizes etc. Out of these possible methods the most suitable and optimum solution is the reduction of complex computations involved [26]. Although other options affect the speed of execution, due to the feasibility of implementation and overall performance enhancement, other methods are not the best options. Improvement of the computational resources can be achieved through switching to higher processors [31] with higher clock rates and large cache memory. However, it is not an overall feasible option as there could be systems with mobile computation platforms with low computation power such as Raspberry pi [28]. It is thus highly expected a versatile solution valid for any system irrespective of the resource availability. Application of hardware accelerators [111] are suitable for devices with peripheral connectivity. The use of GPU devices [73] [74] offer faster computations involved in convolutional as well as dense layers. However, the power consumption [112] being a limiting factor for mobile

devices such as service robots, UAVs etc. In terms of architectural modifications or optimizations, which are related to the model parameters and directly related to the inference. Most relevant modifications are discussed below.

1. Architectural modifications of the Model: the execution time can be reduced significantly by reducing the number of layers [26] and parameters involved during the complex computation steps. Ideally the convolutional layers and hidden layers contribute to the largest inference. However unnecessary reduction of layers may lead to the accuracy of model and thus the selection should be wisely made with trial and error as there's no rule to be followed. Choosing an efficient architecture with guaranteed performance is a wise selection before any convolutional neural network related activity.

2. Model filter pruning: reducing filter parameters such as weights results in reduction of computation burden. If unnecessary filters are present, removal of such filters reduces the model size and thus the overall computations.

3. Kernel optimizations: using optimum kernels compatible with deep learning libraries and hardware specific libraries results in speeding up the process as the optimum computations are selected by the libraries.

The hardware resources can also be changed to enhance the processing speed [31] [76] of a convolutional neural network. Due to the physical component changes it is an expensive option for speed enhancement. However most relevant hardware resource related modifications are presented below.

1. Application of hardware acceleration: rather standalone central processing unit, a graphical processing unit GPU [80] [31] or a tensor processing unit TPU which offers more computation power remarkably reduces the time consumed for processing complex computations involved. The presence of such devices not only speeds up the execution of the CNN but also reduces the time taken during training of the neural network model.

2. Batch processing: During the execution of the network model, processing the data in larger batches reduces the time consumption significantly. It is advantageous in parallel execution on hardware through techniques such as multi-threading and multi-core processing. However, batch processing requires adequate memory allocation which is another expensive option.

3. Dedicated inference engines and hardware: certain hardware platforms are blended with sophisticated inference engines such as NVIDIA [31]. Use of such inference engines which are optimized can accelerate the computations and thereby reduce inference time. Upgrading the computation hardware is also an expensive option where overall performance can be enhanced.

4. Parallelism: Presence of multiple computation devices or multiple CPUs [77] can offer two types of parallelism. Model parallelism, where a large network model can be split into multiple processors and process different sections of the network model concurrently. Data parallelism on the other hand, divides the input data across multiple devices or processor units and processes them simultaneously. Thereby the overall process will be enhanced and thus reduce inference time. However, it is necessary for multiple devices or processors to be available.

2.9 Scientific approaches to enhance the processing speed of CNNs.

There are various attempts made by researchers to enhance the execution speed of a CNN [84] [85] [111] [113–118]. As explained in previous section 2.5 and section 2.8 the root causes for the network model's execution time and how they can be mitigated by model architectural and hardware changes [113] [111] [119]. In this section various attempts made by researchers are discussed, which supports the explanation in previous chapters. Even though hardware resource related modifications are efficient, due to the cost of operation architectural modification has been a major research trend. Dimensional reduction leads to fast processing as there were deduction of computations involved which we described in section 2.8. L.F. Brillet et al [72] demonstrated the effect on dimensional reduction of CNN. Comparison of novel CNN architectures such as YOLO with performance matrices presented in [28–30] [40] [80] [83] [31] [120–122]. In addition to quantization and model parameter pruning techniques to reduce computation complexity, principal component analysis has been adapted. Replacing filters in multiple layer network, allowing reduced weights and thus computations needed less storage resulting overall enhancement on inference time. With respect to the architectural modifications [30] [40] [123] [124], essentially convolutional and fully connected layer modifications, a novel approach of speeding up CNN by image preprocessing and post-processing [73] developed by T. Fukagai et al. Speeding up of the classification process by reducing the exploration of the CNN [75] was developed by J. Pasquet et al where a support vector machine combined to feature maps of the CNN and thereby reducing unnecessary time taken by the CNN. Speed up of CNN used for object detection in real-

time video application [26] by combining region proposal algorithm with sophisticated network model, Faster R-CNN [122] demonstrated by T.M. Phuong and N.N. Diep. A novel approach for fast inference has been proposed by S. Durvasula et al for the application of event cameras. Reduction of computations by looking into consecutive input tensors [25] where only small difference present for input camera and thus reducing large number of floating-point computations. Reducing unnecessary computations had led them to speed up inference up to 1.5 times. Another network model architectural change has been presented by J Pang et al for the fast detection of tiny objects in large scale remote sensing images [78]. It is important to accurately measure the inference latency of a deep neural network. L.Y Zhang et al presented a novel approach for determination of inference by dividing to kernel level [79]. The importance of hardware resources to reduce the inference time has been highlighted by Y. Yue et al for the work in speed up neural network training using supercomputer [76] where the data parallelism performed on multiple processing units. Their results comparatively demonstrate the effect on speed up of the performance as the processing units changed. A. Lavin and S. Gray have extensively discussed the importance of Graphical Processing Unit and optimized libraries developed novel fast algorithm for the convolutional neural networks [30]. A similar approach can be seen in fast convolutional network evaluation [31] by N. Vasilache et al for their comparison of performance profile of convolutional neural networks. Their optimized library cuFFT [31] has demonstrated 1.4 – 14.5 times speed up in CNNs. With the presence of more convolutional layers and dense layers neural networks termed as deep neural networks which offers better generalization capabilities [125]. Comparison of Deep neural network (DNN)s with their performances and proposals for acceleration are presented in [123–127]. Mathematical optimizations and enhancements for deep neural networks are discussed in [81, 82, 112, 128] presenting

shift and accumulation unit which is efficient compared to multiply and accumulate unit. One of the possible methods to speed up the execution of CNN models is optimization of computations involved [112]. Reduction of unnecessary Multiply and accumulate (MAC) operations [129–132] being one of the major research areas in CNN optimization. Efficient feature activation and extraction methods [133–135] and normalization functions [136] [137] are important factors in optimized neural network. Every CNN architecture's common structural element and underlying mathematical explanations are extensively discussed in I. Goodfellow [4]. Another aspect of enhancing classification accuracy by optimizing nodes in dense propagation [138] [6] [139] which overcomes the overfitting. Optimizations in fully connected layers and activations also enhance generalization capacity [140–142] which results in reduction in complex computations. Performance of convolutional neural networks can be improved with optimization of computations as presented by A. Hazarika et al where the computations involved considered as multiply and accumulate (MAC) operations which repeatedly appears across the different layers. Due to the presence of high precision floating point computations on training as well as inference deep neural networks are computationally expensive [128] [143]. A novel approach to reduce computations viewed as multiply and accumulations by reducing large models into smaller models and optimizing weights [128]. Reduction in MAC operations in convolutional layers presented by J Chang et al for speedup deep neural networks resulting in 28% Increase in inference speed [132] within accelerator environment. Table 2.2 presents comparison of different convolutional neural network models along with their learnable parameters and MAC operations for the convolutional and fully connected layers.

Table 2.2: Comparison of CNN Models: Parameters and Operations

Model	**Parameters**	**CONV** (%)	**FC** (%)	**Operations**	**CONV** (%)	**FC** (%)
AlexNet	61.0M	3.8	96.2	725M	91.9	8.1
VGG-16	138.0M	10.6	89.4	15,484M	99.2	0.8
GoogLeNet	6.9M	85.1	14.9	1,566M	99.9	0.1
ResNet-50	25.6M	-	-	3,900M	-	-

The drawback in convolution operation as a dot product of two vectors at the convolution layers [144] [145] was highlighted and the solutions presented for this issue was broadly on two streams software optimizations and hardware optimizations [146]. In terms of hardware speedup of the CNNs inference various approaches have been suggested such as Field programmable gate array (FPGA) based MAC unit designs [145] for higher degree of parallelism. Another approach to speed up inference by reducing off chip memory access [144] while achieving minimal resource utilization. Design of Approximate MAC unit had been presented by Hang Xiao et al for the purpose of CNN acceleration ensuring fast and accurate results. The design was taken into consideration of distribution probabilities of weights and activations and developed on 4:2 compressor and Dadda multiplier [143]. Even though the FPGA based CNN speedup approaches are hardware based and are not relevant for proposed network model design, the weaknesses associated with convolutional neural network were highlighted. One of the approaches for speed up CNNs inference through the model architecture enhancements. Various research focused on parallel implementation on deep networks to reduce the inference time as multiple processors can compute concurrently. Most of the cascaded approaches [147–149] are best suitable for extraction of features concurrently

and thereby reduce the total time for inference. K Wang et al proposed a human face detection based on fast region based convolutional neural network (Fast R-CNN) adapting pRectified linear unit (ReLU) [149] for multistage detectors achieving impressive results. Similar application of cascaded feed forward network can be found in [150] for the application of plant species identification using flower images. R Raid et al have adapted model parallelism as well as data parallelism for biometric identification purposes [151] which presents attempt on adapting cascaded architectures for enhanced performance of CNNs. Another adaptation of Fast R-CNN for cascaded operation by Yu Li et al for the defect detection on fabrics in textile industry [152] with impressive accuracy in defect detection. Cascade R-CNN [153] presented by Z Cai et al have made a significant impact on multi-object detection in real time as compared with other two stage detectors and thus adapted in various research [152] [154] [147]. Both model parallelism and data parallelism are important concepts in CNN training on large datasets. Parallelism leads to effective utilization of computational resources and producing classifications with improved accuracies. However, it does not propose improvement in speed up of inference.

2.10 Lightweight CNN architectures

Lightweight CNN architectures are designed to offer benefits such as reduced computational costs, low power consumption, and a smaller memory footprint, while maintaining competitive accuracy. These features make them ideal for deployment on resource-constrained devices, such as edge devices.

Common design techniques used in the development of lightweight CNN

models include depthwise separable convolutions, inverted residual blocks, and compound scaling. The key principle guiding these designs is to minimize the number of arithmetic operations and memory usage, both of which are critical factors for inference time and energy consumption, especially in embedded environments.

Reducing inference time is one of the key advantages of lightweight designs, which can be attributed to several factors. Depthwise convolution minimizes the number of mathematical multiplications and additions, leading to a decreased computation time. In their work, Sandler et al [98] introduced inverted residuals and linear bottlenecks in MobileNetV2 to enhance feature reuse, resulting in fewer floating-point operations (FLOPs) and reduced memory access. The following version, MobileNetV3, developed by Howard et al [155] utilized neural architecture search (NAS) and lightweight attention modules to further optimize the balance between latency and accuracy for mobile CPUs. EfficientNet, created by Tan and Le [156] demonstrated that careful compound scaling of depth, width, and resolution can produce more accurate models that are computationally efficient compared to standard scaling methods. A smaller model with fewer parameters leads to reduced memory traffic, which in turn shortens inference time.

Considering energy consumption, lightweight designs often contribute to greater energy savings due to their reduced computational needs, minimized memory movement, and compatibility with low-power hardware. Yi et al [157]. demonstrated that depthwise accelerators can lower energy consumption, achieving an energy efficiency of 11.13 TOPS/W. They also noted that reduced intermediate data access leads to decreased latency.

Nagel et al [158] illustrated how quantization enhances energy efficiency,

showing that INT8 quantization significantly lowers inference power consumption while causing minimal accuracy degradation, making it suitable for embedded environments. Husom et al [159] emphasized the importance of quantization, which notably reduces energy consumption in large language models tested on Raspberry Pi.

Rossi et al [160] adapted depthwise convolution with early downsampling, resulting in a lightweight model with reduced complexity while preserving high accuracy, tested on low power edge devices such as the Jetson Nano and Raspberry Pi. This showcases the effectiveness of lightweight CNN models in energy constrained environments. These results show that lightweight CNNs reduce inference latency and energy consumption, enabling real-time use in power constrained applications like service robots, IoT devices, and mobile systems.

2.11 Energy efficient CNN architectures

Energy efficient Convolutional Neural Networks (CNNs) are a significant advancement in deep learning, making it possible to deploy intelligent vision systems on battery powered and resource constrained edge devices. These devices include smartphones, microcontrollers, drones, wearables, and Internet-of-Things (IoT) sensors. As demand for on device artificial intelligence increases driven by the need for low latency, privacy, and offline operation the energy cost of inference has become a primary design constraint. Recent studies emphasize optimization techniques such as pruning, quantization, lightweight architectures, and hardware-aware design as essential components of modern optimization strategies.

Numerous studies have demonstrated the effectiveness of various methods

in reducing computational costs, inference latency, and power consumption on constrained devices. For example, Salar et al [161] showed a significant energy enhancement of 72% with a novel MAC layer in their proposed architecture for edge devices, all without sacrificing accuracy. Meanwhile, Ju et al [162] introduced TripleNet, a lightweight CNN architecture tested on the CIFAR-10 and Street View House Numbers (SVHN) datasets. This architecture achieved a 15-30% reduction in latency compared to standard networks like MobileNet and EfficientNet when tested on a Raspberry Pi device. Similarly, Sui et al [163] proposed a hardware friendly pruning and quantization framework that resulted in substantial energy savings on FPGA platforms, highlighting the cross platform relevance of such optimizations. Naveen and Kounte [164] emphasized that their proposed three-stage pipeline, which incorporates pruning and quantization, has led to a 22.7% improvement in latency and a 29.4% increase in energy efficiency when applied to the YOLO V3 model. This approach is also suitable for deployment on IoT edge devices. Liu et al [165] introduced a lightweight model for feature extraction in visual simultaneous localization and mapping systems, achieving an impressive 88.3% reduction in model size, which resulted in a 22% increase in energy efficiency compared to state-of-the-art models.

Additionally, while discussing the advantages of model compression, Jayasimhan and Pabitha [166] pointed out a significant drawback of current pruning strategies: the permanent loss of weights due to filter pruning. They addressed this issue through a two-phase pruning technique, which achieved a 57% reduction in FLOPs with only a 0.08% degradation in accuracy.Weight quantization is a technique that aims to the computational complexity of convolutional neural networks (CNNs) while maintaining accuracy. However, the value locality of quantized filter weights can lead to computational

redundancy. Ghanbadri and Modarressi [167] proposed a novel architecture that eliminates this redundancy. Their approach has achieved energy efficiency improvements of 10.5% to 20.9% and latency reductions of 13.4% to 41.6% compared to standard CNN models.

In addition to the benefits of energy efficient CNNs mentioned above, there are challenges and trade-offs that must be addressed, such as accuracy loss, design complexity, and deployment issues. These challenges require rigorous experimental analysis. By designing a model that aligns with the target hardware constraints such as memory bandwidth, precision support, and compute capacity, energy-efficient CNNs can achieve real-time performance with minimal power consumption. This often results in lower energy usage per inference, enabling sustainable operation on edge devices.

2.12 Benchmarking and evaluation matrices of CNNs

Convolutional Neural Network (CNN) benchmarking is a systematic process for evaluating and comparing deep learning models. The entities considered in these evaluations are the assessment metrics. Benchmarking and evaluating CNNs require a comprehensive approach that takes into account both algorithmic accuracy and system-level efficiency, especially for real-time applications on edge and embedded platforms.

Standard accuracy metrics, such as Top-1 and Top-5 classification accuracy, are fundamental indicators of predictive performance. In tasks involving imbalanced datasets or detection, metrics like precision, recall, and F1-score are often used to better capture error sensitivity [168]. However, accuracy

alone does not suffice for practical implementations. Latency, defined as the inference time per sample, is critical in time-sensitive applications like robotics and autonomous driving, where delays of just a few milliseconds can negatively impact system responsiveness [169].

Throughput, commonly measured in frames per second (FPS), evaluates the system's capacity under continuous load and supports latency assessment, particularly in high-volume scenarios like video analytics. Energy consumption and memory footprint are also crucial on edge devices, where battery limitations and restricted RAM/storage impose significant constraints. Research has shown that optimized CNNs can achieve up to 45% lower energy consumption on devices like the Raspberry Pi and Jetson Nano without a substantial loss in accuracy [170].

Further metrics, such as model size (parameter count), computational complexity (FLOPs or MACs), and quantization levels (FP32, FP16, INT8), are often reported as proxies for deployment costs. However, these metrics do not always correlate directly with real-world latency due to interactions between hardware and software [171].

Therefore, comprehensive benchmarking requires multi-dimensional evaluation: accuracy for effectiveness, latency and throughput for responsiveness, and energy and memory considerations for sustainability, along with trade-off analyzes across these dimensions. Recent studies highlight the importance of reporting device specifications, batch sizes, and optimization strategies (e.g. pruning, quantization, cascaded inference) alongside metrics to ensure reproducibility and fair comparisons [172]. In addition to these evaluation measures, percentile tail latency, first introduced by Theodoros et al [173] measures the worst delays experienced by a fraction of inputs, as stated in [174]. Software overhead is another important met-

ric, as the overhead from frameworks, libraries, drivers, and data transfer can significantly impact inference time and energy consumption. As mentioned in [175], benchmarks often compare different frameworks. The characteristics of the platform are also crucial, as they affect all metrics. This includes factors such as CPUs, GPUs, Neural Processing Units (NPUs), clock speeds, and memory bandwidth. The selection of these evaluation metrics ultimately depends on the specific requirements of the application and any design constraints.

2.13 Cascaded CNN architectures

Cascaded convolutional neural network is an architecture which stacks more than one CNN in a sequential manner. Each CNN in the stack refines the predictions of the previous network model. Cascaded CNN architectures are particularly useful in hierarchical processing and refinement which produces more concise results as an artificial neural network. Cascade CNNs have been used in various applications such as object detection [147] [150] [153] [154] facial landmark detection [149] [151] [176], medical imaging [177] [178], pedestrian detection [179]. Feature pyramid network has been adapted in improved cascade R-CNN [152] by Y. Li and Z.W Wang for defect detection in textile industry. Considering the earlier attempts for the cascaded CNN architectures, M.W Hirsch [20] has described extensively the importance of the convergence of the cascaded neural networks with supported mathematical elaboration to the concept of cascading of the network. J. Huang et al [16] has presented adaptation of cascaded CNN for automatic modulation identification in communication systems with an impressive classification accuracy. Application of recurrent neural networks in cascaded architecture presented in [17] and [18]

highlighting the efficiency and robustness. Cascaded feature extraction on Bi-LSTM architecture [17] for the purpose of emotion identification based on EEG signals conducted by Y. Luo et al highlighting the accuracy of the results. S-Y. Sun et al [19] have successfully demonstrated the applicability of memristor based neuromorphic computation units in cascaded form. Y. Ren and B. Wang [21] have presented 3-layer CNN as a weak classifier assisted with Adaboost algorithm where cascade in stages producing better performance compared with linear SVM classifiers in terms of accuracy and detection speed. T. Shen and D. Zhu [22] have demonstrated the effectiveness of layered cascade in constructive neural networks. A.J. Dhanaseely et al [23] have adapted cascaded feed forward neural network for face recognition system where its performance analysis clearly stating improved detection accuracy compared with standard feed forward neural network models. Research work conducted by Z. Lu and C. Tang [24] highlighted the importance of training samples in cascaded back propagation networks. The generalization capacity enhancement from the cascading architectures further explained by Z. Cai and N. Vasconcelos [153] work on RNNs where cascaded architecture has overcome the problem of overfitting. It also highlighted the applicability of cascaded architecture to many object detection models. Impressive generalization accuracies [16], fast inference [21] and applicability to object detection architectures [153] [180] influenced this research work to apply two stage cascaded architecture for further analysis.

2.14 Advantages, Challenges and Applications of Cascaded CNN architectures

Cascaded Convolutional Neural Networks (CNNs) provide notable advantages over traditional CNN architectures, especially in real-time or resource-limited environments like embedded systems and edge devices. The key idea behind a cascaded architecture is to process inputs through multiple stages, enabling the system to make early and confident predictions. This staged approach offers several significant benefits, which are discussed below.

1. Reduced inference time: One of the most significant advantages of cascaded CNNs is their ability to reduce average inference time. A large portion of real-world data comprises "easy" inputs, which can be accurately classified by the initial lightweight stages of the network without engaging the deeper layers. As a result, the system can process the majority of inputs quickly, with only a small subset requiring full network evaluation. According to the results presented in [181], the adaptive computation mechanism can increase throughput by over 50% in some cases compared to single deep CNN baselines, all without a significant loss in classification accuracy.

2. Resource efficiency: By avoiding redundant computations for simple inputs, cascaded CNNs significantly lower the number of multiply accumulate operations and the associated memory traffic. This reduction leads to decreased energy consumption, which is especially beneficial for edge devices and mobile systems that operate under strict power constraints. Studies have shown that cascaded architectures can reduce energy usage by 30-40% compared to single stage

networks while maintaining similar accuracy levels. This makes them an attractive option for deployment in battery-powered devices, such as guidance robots and IoT cameras.

3. Improved accuracy latency trade-off: One significant advantage of cascaded CNNs is their ability to establish tunable confidence thresholds, allowing for a balance between accuracy and latency. These confidence thresholds can be adjusted at each stage to determine whether a sample should exit early or continue to the next stage. In scenarios that require high responsiveness, the thresholds can be tightened to maximize speed, which may result in a slight decrease in accuracy. However, in tasks where accuracy is critical, the thresholds can be relaxed to ensure that more samples progress to deeper stages. Alexandros et al. [73] emphasized that this runtime adaptability gives system designers the flexibility to dynamically optimize performance according to application requirements or hardware constraints. Additionally, Alexandros et al [182] pointed out the value of variable operating points, noting that they make cascaded systems particularly useful for changing requirements, such as switching between battery-saving modes and high accuracy modes. Such adaptive policies can be employed to meet service level objectives in a dynamic manner. Zhaowei and Nuno [183], [184] presented a two-stage cascaded recurrent model that significantly improved detection quality while minimizing overfitting and the mismatch between inference time and quality.Sun et al. [185] reported that their cascaded faster recurrent CNN model outperforms the state-of-the-art faster R-CNN model in classification accuracy.

4. Suitability for edge environment deployment: Cascaded CNNs are particularly effective in heterogeneous environments where compu-

tational tasks are distributed across various hardware units. The lightweight initial stages can be executed on a CPU or a low-power accelerator, utilizing parallel processing elements that can handle multiple threads. This setup speeds up the overall inference process since the cascaded layers can be executed across separate CPU cores. Depending on the load and energy budget, thresholds can be adjusted to enable early exits with greater confidence while maintaining accuracy. Nicholas et al. [186] demonstrated a similar approach in distributed inference using Raspberry Pi, highlighting the edge compatibility of the cascaded architecture.

While the benefits of the cascaded CNN architecture are evident, it also presents challenges that need to be addressed. These challenges range from algorithmic design to considerations for systems operating in real-time and resource-constrained environments.

1. Confidence threshold and error propagation: A significant challenge in cascaded CNNs is determining the appropriate confidence thresholds for early exits. If these thresholds are set too aggressively, the system may misclassify inputs by exiting too soon, leading to errors that later stages cannot correct. However, if the thresholds are set too conservatively, the advantages of cascading are diminished, as unnecessary inputs are sent to deeper and more costly stages. Achieving a balance between accuracy and efficiency through threshold calibration is a complex task that requires careful tuning and validation on representative datasets.

2. Increased training and complexity: Unlike single CNN models, cascaded architectures require careful design across multiple stages, including consideration of their structure, depth, and precision. Each

stage can be trained either independently or jointly; however, achieving complementary learning among them presents challenges. Joint optimization may lead to instability during convergence, whereas independent training could result in redundancy or performance imbalances between the stages. Additionally, training cascaded models often requires extra labeled data to calibrate exit policies, which increases the development overhead.

3. Challenges in Hardware Implementation: While cascades are conceptually efficient, their irregular execution flow characterized by conditional early exits creates challenges for hardware accelerators like GPUs and FPGAs, which are optimized for batch-parallel workloads. Divergent execution paths resulting from confidence-based branching lead to reduced efficiency in hardware utilization. For instance, on FPGAs, implementing multiple stages with varying precision levels complicates resource allocation and pipeline scheduling.

Cascaded CNN architectures are well-suited for a variety of real-world applications because of their benefits, which include efficiency, accuracy, and scalability. Their staged structure enables early exits for simpler cases while allowing deeper processing for more complex inputs. This makes them particularly effective for tasks that require real-time performance on resource-constrained devices. Below are some notable applications of cascaded CNN architectures.

1. Real-time robotic and guidance systems: are one of the most important applications of cascaded CNN architecture. One of the earliest and most impactful applications of cascaded CNNs is in object detection. Cascade based approaches such as Cascade R-CNN progressively refine region proposals using multiple stages with increasing

Intersection over Union (IoU) thresholds, resulting in significantly improved detection accuracy compared to single-stage detectors [187]. Similarly, cascaded face detection models, including the Multi-task Cascaded CNN (MTCNN), achieve robust detection under challenging conditions such as occlusion and varying illumination by filtering easy negatives in early stages and refining detections in later ones [188].

2. Face recognition and biometrics: Cascaded Convolutional Neural Networks (CNNs) are commonly used for tasks such as face alignment, recognition, and liveness detection. The cascaded design enables early rejection of non-face regions through lightweight classifiers [188], while deeper models manage more detailed recognition tasks. This strategy enhances efficiency in large scale biometric systems [189], including surveillance and access control, where rapid and reliable face identification is essential.

3. Medical Imaging and Diagnosis: In healthcare, cascaded Convolutional Neural Networks (CNNs) are utilized for various tasks, including tumor detection, lesion segmentation, and disease classification. The initial stages of these networks quickly filter out normal cases or those that are easy to classify, while the later stages focus on providing a more detailed analysis of suspected abnormalities. This method not only accelerates the diagnostic process but also lightens the workload on resource intensive imaging systems. Remarkable accuracy and performance have been demonstrated in applications across dermatology [190], ophthalmology [191], and radiology [192].

4. Surveillance Systems: In large-scale video analytics, cascaded Convolutional Neural Networks (CNNs) facilitate efficient processing by filtering out easy background frames in the initial stages. This allows

only suspicious or ambiguous regions to be forwarded for deeper analysis. This hierarchical filtering mechanism helps surveillance systems maintain real-time performance by directing resources toward relevant events. Notable applications of cascaded CNNs include object detection in surveillance [193] and anomaly detection in surveillance video [194].

Cascaded CNN architectures are utilized in various fields, including computer vision, speech processing, healthcare, and edge computing. Their capacity to balance accuracy, latency, and energy efficiency makes them especially suitable for real-time applications that require resource management and high throughput. These qualities contribute to their increasing adoption in both industry and research settings.

2.15 Chapter Summary

This chapter provided a comprehensive review of the existing literature related to the world of convolutional neural networks with a particular focus on the CNN's execution, layers, mathematical operations. It began with basics of convolutional neural networks architectures, evolution efficiency, limitations and various types factors affecting inference speed. subsequently the chapter examined the inference time enhancement methods, their applications, and performances. Various researches focusing on inference speed improvements have been discussed with the mathematical operations involved in CNN architectures and their optimizations. The literature review identified the gaps in current research including computation complexity over the increasing depth of the network model. The chapter concluded by justifying the need of computational optimizations

to enhance the inference speed of a CNN model. light weight CNNs involved lesser number of complex computations to produce its generalization compared to lengthy dense networks offering similar accuracy. These findings formed the foundation for the methodology proposed in chapter 3.

CHAPTER 3

Methodology

Contents

3.1 Introduction to the Chapter **64**

3.2 Motivation to the novel approach **65**

3.3 AlexNet overview **67**

3.4 Proposed Cascaded CNN architecture **70**

3.5 Main network architecture **74**

3.6 Secondary network architecture **76**

3.7 Major CNN layers and operational principles **77**

3.7.1 Convolutional Layer 77

3.7.2 Activation function 86

3.7.3 Normalization of the intermediate results 91

3.7.4 Pooling layers 93

3.7.5 Dropout layer 95

3.7.6 Fully connected layer 101

3.7.7 SoftMax layer 103

3.7.8 Depth concatenation layer 105

3.8 Methodology **106**

3.8.1 Computations involved in Convolutional layer . 107

3.8.2 Determination of complex computations in Convolution layer 109

3.8.3 Importance of analysis of feature maps 112

3.8.4 Computations in secondary network 112

3.8.5 Computations from the ReLU activations . . . 115

3.8.6 Computations from Maxpool layer 115

3.8.7 Computations from Fully connect layer 117

3.9 Implementation **118**

3.9.1 MATLAB implementation and deployment . . . 121

3.9.2 Testing methods 126

3.9.3 Inference time measurement 126

3.9.4 Execution speed measurement using profiler tools 128

3.9.5 Hardware testing 130

3.10 Proof of the concept **134**

3.10.1 Determination of computations involved in AlexNet convolutional layers 135

3.10.2 Determination of computations in AlexNet ReLU layers 140

3.10.3 Determination of computations in AlexNet Maxpool layers 141

3.10.4 Determination of computations in AlexNet Fully connected layers 143

3.10.5 Execution time measurement 146

3.11 Chapter Summary **150**

3.1 Introduction to the Chapter

This chapter outlines the research methodology and system design framework used to develop the proposed cascaded convolutional neural network (CNN) model for real-time image classification on edge devices. The methodology aims to achieve an optimal balance between classification accuracy and inference speed, while addressing the computational limitations of embedded platforms like the Raspberry Pi. The research process integrates algorithmic design, architectural optimization, and empirical evaluation to create a lightweight yet high performing model that is suitable for mobile and robotic applications. The proposed approach's overall workflow, illustrated in Figure 3.2, outlines the sequential research stages of this study. The process begins with data acquisition and preprocessing, followed by the design and implementation of the main Convolutional Neural Network (CNN), which conducts initial classification using a multi-branch convolutional architecture. A confidence threshold mechanism is employed to determine whether the prediction is reliable enough or if it requires further refinement by a secondary CNN subnetwork. This cascaded structure optimizes computational resource use by activating deeper layers only for uncertain predictions. Finally, the system's output is sent to a controller module for downstream decision making, allowing for real-time integration with intelligent robotic or embedded systems. The methodology used in this research is primarily experimental and includes both simulation based testing and hardware level validation. The Convolutional Neural Network (CNN) model was trained and tested using standard benchmark datasets. It was then deployed on an edge computing platform to assess its real-world performance regarding accuracy, inference time, and energy efficiency. To improve model compactness and responsiveness, several op-

timization techniques were systematically applied, including efficient layer design, hardware aware design, layer fusion, and pruning.

3.2 Motivation to the novel approach

Before explaining the new network model in detail, it is important to discuss the necessity of a new network model which outperforms the standard feed forward neural network. Considering the number of computations involved during the execution of the network model, which is directly related to the execution time consumption, thus execution speed. Therefore, it is necessary to study the computations involved in standard feedforward neural networks. As explained in chapter 2, the new network model contrasts with AlexNet [2]. Referring to the explanations in chapter 3.6, the number of learnable parameters can be listed as shown in Table 3.1 below.

Table 3.1: Learnable parameters of Alexnet

Layer	**Learnable parameter**
Convolution 1	34944
Convolution 2	307456
Convolution 3	885120
Convolution 4	663936
Convolution 5	442624
Fully connect 1	37752832
Fully connect 2	16781312
Fully connect 3	4097000
Total	60961128

Based on the learnable parameters and the number of computations involved, the complete execution time was calculated. The practical value of

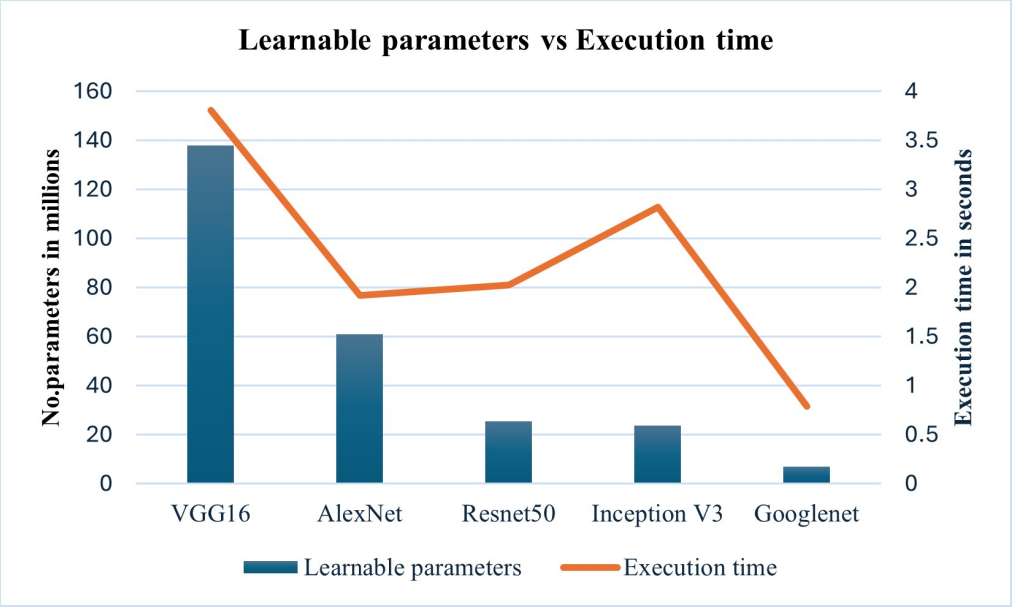


Figure 3.1: Comparison of Learnable parameters with execution time

AlexNet execution was also determined using inference time measurement, one of the testing methods explained in chapter 3.6, and the values are presented in Table 3.2 below.

Table 3.2: Comparison of Network Models: Parameters and Execution Time

Network Model	**No. of Learnable Parameters (in millions)**	**Execution Time (seconds)**
VGG16	138	3.804877
AlexNet	61	1.919307
ResNet-50	25.5	2.026292
Inception-v3	23.8	2.818361
GoogLeNet	6.9	0.788366

With the information gathered for the neural network's complex computations involved and number of learnable parameters present it was then considered to determine the available network models and contrast their performances over the specifications as presented in the Figure 3.1.

To make a comparison on learnable parameters verses execution time, several neural network models which were widely used were selected such as GoogLeNet, AlexNet, Resnet50, Inception v3 and VGG16. From Figure 3.1 it is much clearer that the time consumed during the execution of the neural network model was reduced as the learnable parameters were reduced. The model VGG16 offered the longest time to execute the network and produce classification output while GoogLeNet offered the least execution time with least number of learnable parameters. Although the inception v3 model offered slightly more time compared to AlexNet and resnet50, the tendency of reduced execution time over deduction of learnable parameters can be seen. It was highlighted that Google net offered the least execution time as compared with AlexNet. Inception module architecture has led the GoogLeNet with variable kernel sizes for the reduction of learnable parameters and thus complex computation which indirectly results in fast execution. This result motivated this research to develop a cascaded neural network model with minimum number of learnable parameters.

3.3 AlexNet overview

With the inspiration from the shallow network architectures, AlexNet is one of the earliest convolutional neural network models which introduced the concept of deep network design while achieving significant recognition from the scientific community over its performances. Ability to classify large number of object classes over millions of datasets from which it was trained presented fair enough accuracy compared to the other models. AlexNet [1] developed by Alex Krizhevsky et al. originally consists of eight computation layers where five convolutional layers and three fully connected layers which contribute to the computation parameters. In ad-

dition to the computational layers, there are image input layers for pre-processing of the input samples, activation functions, batch normalizations and dropout layers present which doesn't contribute for the computation parameters. AlexNet adapted intermediate activations such as Rectified and linear unit (ReLU) proposing better solutions for vanishing gradient problem. Demonstration of the effectiveness of image classification applications have brought the attention of the researchers and thus popularizing the new era of deep neural networks. It has motivated the development of more efficient, sophisticated, and robust neural network architectures revolutionizing deep neural networks. The massive success of AlexNet can be elaborated on the following perspectives as explained further below. Simple yet efficient architecture AlexNet is explained as an eight layer convolutional neural network since it consists of eight computational layers. Instead of a single convolutional layer, a greater number of computational layers are present allowing the network model to learn more features from the dataset on which it has been trained thus producing more reliable output. Application of ReLU activation functions, one of the great issues associated with shallow networks with many layers was the vanishing gradient problem associated with traditional activation functions such as sigmoid where the gradients of the loss function approach zero resulting the network model too difficult in training. Shallow networks with lesser number of convolutional layers doesn't encounter with such difficulty but with a greater number of layers involved such as in AlexNet, training becomes much more difficult. However, with the use of rectified and linear units, a clear solution to the vanishing gradient problem allowing more efficient training on deep neural networks has been successfully produced. Local Response Normalization (Local response normalization (LRN)) in AlexNet architecture batch normalizations have been adapted in between consecutive hidden layers to normalize the responses from neighboring channels.

This has resulted in enhanced generalization and thus robustness into the network model. Parallel processing and availability of data, AlexNet has been designed to train on multi-GPU environment thus enabling parallel processing. Therefore, reducing the network training time significantly and producing more efficient network model. Since the network's architecture with model parameters such as weights and biases along with the layer details publicly available, motivated more research to further experiment on this architecture and apply them on without any limits and restrictions. This has led to the success of later network models inspired by the AlexNet model. Since the aforementioned advantages have led to the key success of the network model and inspiration to the later efficient and optimized network models it is also good to know the limitations associated with this network. Even though it offers more benefits the limitations have led us to develop the proposed network model. Some of the key limitations are discussed below which motivated the proposed network. Computational complexity, AlexNet architecture consists of five convolutional layers and three fully connected layers. Each of these layers involved in complex mathematical computations which resulted in a larger number of learnable parameters. The massive number of complex computations require large computation power essentially a Graphical Processing Units (GPU) rather central processing units (CPU) and more sophisticated memory to store the results including the intermediate results. Therefore, Alexnet required sophisticated computer resources to conduct experiments and thus limited researchers could work on further improvements. Large number of learnable parameters, as explained earlier, the AlexNet model has a large number of learnable parameters from each convolutional layer and fully connected layer. Especially a significantly large number of learnable parameters were introduced by the three fully connected layers. This resulted in overfitting over training of the network model with lesser number of training data.

Therefore, direct proportion could be observed over number of parameters and number of training dataset to produce more generalized result. Overfitting, as mentioned earlier, AlexNet model computes large number of learnable parameters which require sufficiently large number of training datasets and thus susceptible to overfitting when limited dataset used for training. When overfitting occurs as the network trained over lesser number of data, the model tends to learn too well on the fewer data, including the noise and certain parts such as outliers which reduces the generalization as the new data is fed for the purpose of classification. Focusing on the above-mentioned limitations associated with AlexNet motivated this research to overcome these issues and produce better results out of an efficient network model.

3.4 Proposed Cascaded CNN architecture

The proposed CNN architecture comprises two stages: the main network and the subnetwork. To clarify the architecture, a flowchart is provided in Figure 3.2. This flowchart illustrates the complete operational pipeline of the proposed cascaded convolutional neural network (CNN) model, which is designed to achieve real-time and efficient image classification on embedded or edge-based systems. The model operates within a two stage framework, with each stage specifically optimized to balance inference speed, memory usage, and classification accuracy. Input Data Acquisition and Pre-processing: The process starts with image acquisition, during which raw image data is captured from an input source, such as a camera or sensor system integrated into a mobile or robotic platform. In the input pre-processing stage, essential operations are performed, including image resizing, normalization, and noise reduction. These steps ensure a consis-

tent data representation for the CNN model. The preprocessed image is then sent to the feature extraction pipeline. main CNN Network (primary feature extraction and classification stage): The pre-processed input undergoes initial processing by the main CNN, which extracts hierarchical spatial features through multiple convolutional layers. Conv1 to Conv5: These layers progressively extract features from low-level to high-level within the input data. Each layer enhances the depth of feature representation while reducing the spatial resolution. Conv1 and Conv2 primarily capture basic edge and texture information. Conv3 to Conv5 focus on more complex patterns at the object level and semantic structures. After Conv5, the architecture splits into parallel convolutional branches to represent features at multiple scales. Conv6, Conv7, Conv8, and Conv10: These four layers operate concurrently on different receptive field sizes, enabling the extraction of features at multiple abstraction levels.

Conv8 to Conv9 and Conv10 to Conv11: Two sub-branches extend vertically downward to enhance deeper feature learning from the outputs of Conv8 and Conv10, respectively.

The outputs from all parallel branches (Conv6, Conv7, Conv8, Conv9, Conv10, Conv11) are merged through depth concatenation to produce a unified feature map that integrates multi-level and multi-scale information.

Fully Connected Layer and Confidence Threshold Evaluation: The concatenated feature map is input into a fully connected (FC) layer, which transforms the high dimensional spatial features into a vector representing classification probabilities. A confidence threshold mechanism is then applied to evaluate the reliability of the predictions. If the classification confidence exceeds the defined threshold, the output is accepted and forwarded directly to the Final Output Interpreter. However, if the confidence

falls below the threshold—indicating uncertainty or a potential misclassification the image is redirected to a secondary CNN network for further refinement. Secondary CNN Network (Refinement Stage): The secondary CNN (snConv1 – snConv5) serves as a fine-grained classifier aimed at improving prediction accuracy for complex or ambiguous samples. This stage employs a similar convolutional structure but with fewer layers and optimized filters to reduce computational costs. The outputs from snConv1 through snConv5 are processed by a fully connected layer, which generates the final class scores. This design ensures that only challenging cases require additional computation, thus decreasing the average inference time while still maintaining high classification performance. Output Interpretation and Control Integration: The Final Output Interpreter merges results from both the main and secondary networks to generate the final classification decision. This output is then sent to the next controller module, which could be linked to a decision-making or actuator control unit in an embedded or robotic application. This integration allows for real-time system responses, making the proposed architecture highly suitable for guidance service robots and other intelligent edge devices.

In summary, the proposed cascaded CNN flowchart describes an adaptive two stage image classification pipeline. The main network enables rapid and efficient processing for the majority of inputs, while the secondary network is activated selectively to address uncertain or challenging cases. This cascaded design significantly reduces computational redundancy, optimizes inference time, and enhances overall accuracy, providing an ideal balance for embedded AI systems.

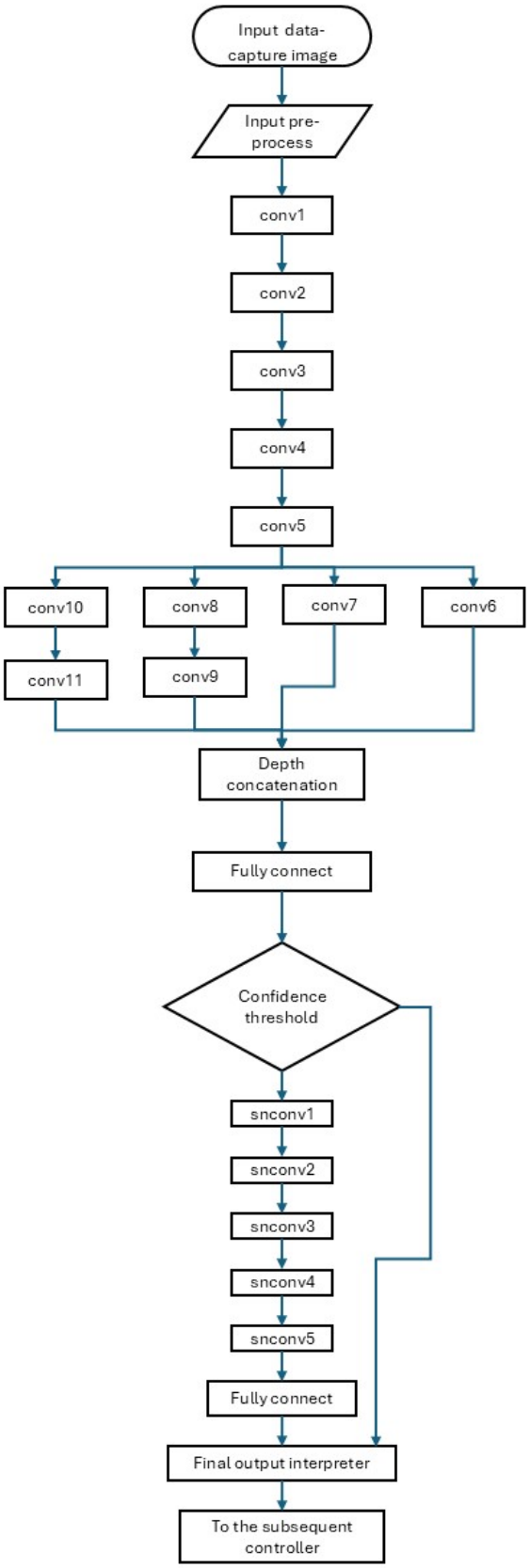


Figure 3.2: Flowchart of the proposed cascaded CNN architecture

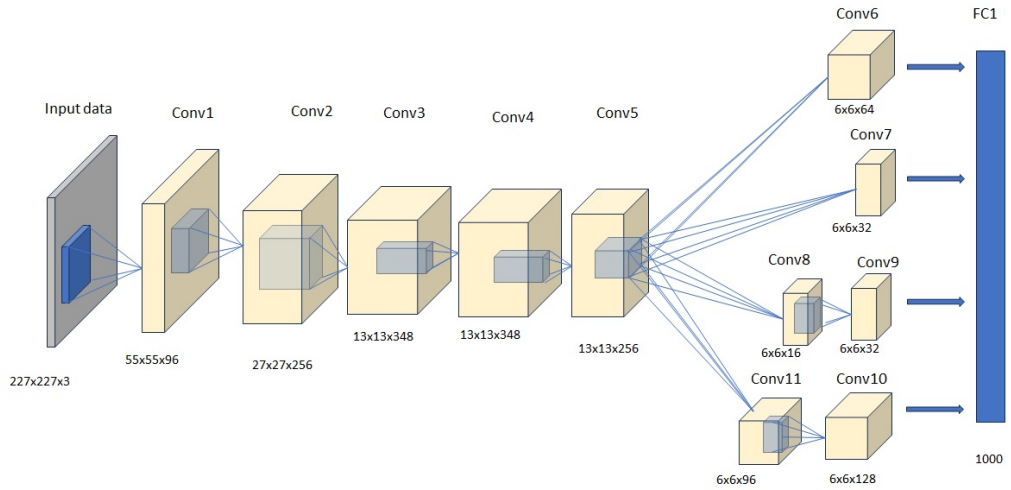


Figure 3.3: Main network architecture

3.5 Main network architecture

The proposed architecture is explained in the form of two distinct models. Main network model is explained in new network architecture 1 and the sub network in new network architecture 2. The main network model acts as the primary classification source, where the object of interest is broadly classified into a group of classes from its extracted features. Main network model is based on feedforward neural network architecture which offers significant accuracy for trained dataset. The main network essentially consists of both sparse and dense connectivity to ensure feature maps extracted from given input image passed through every layer and successfully connected to last fully connected node having all the other intermediate operations such as activations and linearization etc.

The main network consists of 35 layers and its arrangement is presented in Figure 3.3 It consists of twelve learnable layers where 11 convolutional and one fully connected layer. Starting with a preprocessing layer which accepts three channel RGB image of size 224 x 224 will be passed down to the network. There are 11 Convolutional layers of different kernel sizes are

used in this network where the first convolutional layer with considerably large kernel size of 11 x 11 with stride of 4 producing 96 feature maps which are activated by rectification and linearization unit ReLU where the activated feature map will be used in next stage of the neural network. The kernel size for the initial convolutional layer was selected to be large in order to accommodate more information as the feature maps passed down through the layers of the network, the kernel sizes are reduced. The main network architecture's specifications are listed in Table 3.3 below. To understand the importance of processing time it is necessary to analyze the learnable parameters, and their specifications while focusing on their outputs.

Table 3.3: Main network model specifications

Layer	**Kernel**	**Filters**	**Strides**	**Output**	**Learnable Parameters**
Conv1	11 × 11	96	4	55 × 55 × 96	34,944
Conv2	5 × 5	256	1	27 × 27 × 256	307,456
Conv3	3 × 3	384	1	13 × 13 × 384	885,120
Conv4	3 × 3	384	1	13 × 13 × 384	663,936
Conv5	3 × 3	256	1	13 × 13 × 256	442,624
Conv6	1 × 1	32	1	6 × 6 × 32	8,224
Conv7	1 × 1	64	1	6 × 6 × 64	16,448
Conv8	1 × 1	96	1	6 × 6 × 96	24,672
Conv9	3 × 3	128	1	6 × 6 × 128	110,720
Conv10	1 × 1	16	1	6 × 6 × 128	4,112
Conv11	5 × 5	32	1	6 × 6 × 32	12,832

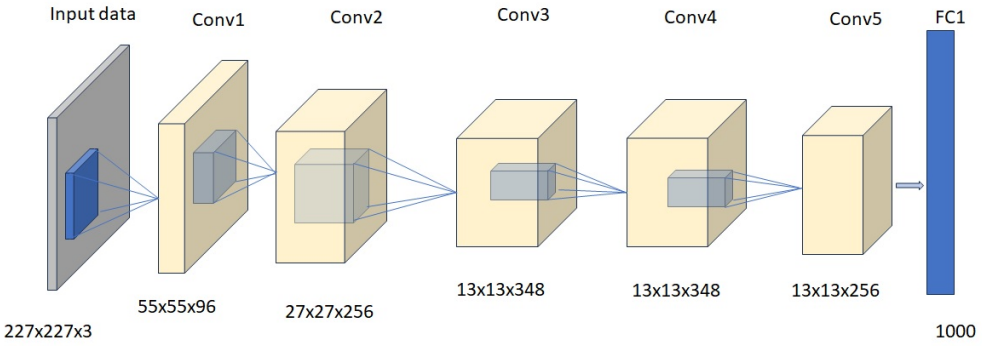


Figure 3.4: Secondary convolutional neural network architecture

3.6 Secondary network architecture

The proposed architecture consists of two sections as described in chapter 3.2. The secondary network model designed was based on the feed forward neural network architecture. Inspired by shallow network architecture [1] [2] the sub network model was developed where it consists of 20 layers with minimum learnable parameters. The six learnable layers where five convolutional layers and one fully connect layer contribute for the reduction of computations for the identification of the same object. Neural network's layer arrangement is presented in Figure 3.4.

As shown in Figure 3.4 the image input of size 227 x 227 x 3 is applied to the input data layer which preprocesses the input and generate feature map as specified to be used in the 2D convolutional layer. With a kernel of size 11 x 11 applied over the image thus producing feature map of size 55 x 55 of depth 96. Since this intermediate feature map is being used by the subsequent layers, it is necessary to apply an activation function to determine which features are to be taken and dropped by the others. The rectification and linearization unit was used for the activation function throughout the implementation as it offers linear relationship. Kernel size of 5 x 5 and 3x3 were applied for the subsequent convolutional layers to

generate the intermediate feature maps. Even though the kernel sizes were reduced, the depth was maintained in following convolutional layers to retain the object information without loss. The subnetwork's specifications are listed in Table 3.4.

Table 3.4: Specifications of secondary network model

Layer	**Kernel**	**Filters**	**Strides**	**Output**	**Parameters**
Conv1	11×11	96	4	$55 \times 55 \times 96$	34,944
Conv2	5×5	256	1	$27 \times 27 \times 256$	307,456
Conv3	3×3	384	1	$13 \times 13 \times 384$	885,120
Conv4	3×3	384	1	$13 \times 13 \times 384$	663,936
Conv5	3×3	256	1	$13 \times 13 \times 256$	442,624

3.7 Major CNN layers and operational principles

An overview of the network architecture was presented on chapter 3.4 and 3.5 respectively for a better understanding of the cascaded network model. However, it is important to present the building blocks of the network and working principles of each of them. In this chapter important layers from which the network consists of are explained in detail with their mathematical explanation of operation.

3.7.1 Convolutional Layer

Convolutional neural networks also known as convnets [5] [8] is neural network type specialized in processing of two dimensional data such as images demonstrating an astounding success in practical applications. Although

designed to work with two dimensional data, convolutional neural networks can be used with one dimensional data such as audio signal which varies with time are effectively capturing temporal patterns and three dimensional data such as volumetric images or video in medical imaging analysis [86] impressively capturing spatial features across the three dimensions. The convolution layer is one of the fundamental building blocks of any convolutional neural network where most of the significant computations take place. Convolutional layer involved mathematical operation, convolution over the input image producing an output for following consecutive layers. Convolution operation is application of matrix multiplication of input with a filter, also known as kernel which produces map of activations referred to as a feature map highlighting the features detected from the input image. Figure 3.5 refers to the operation principle of convolution where the kernel, a two dimensional array of weights is moved across the input image which is an array of two-dimensional data producing convolved feature map from the corresponding pixels of the input image. Multiplication performed between input data and weight kernel is a dot product with element-wise multiplication where the products are summed resulting in one value.

As shown in Figure 3.5 the kernel is moved across the input image with predefined steps known as strides ideally from left to right and top to bottom. The parameters associated with the kernel are generated by the algorithm decided on the loss function which include various steps such as initialization, training, back propagation, gradient decent optimization etc. most of the libraries implement the convolution operation by means of cross-correlation avoiding kernel flipping to maintain commutative property associated with convolution. The weight kernel operation results in intermediate output as shown in Figure 3.6 from LeCun et al [5] clearly presents the product of input pixels multiplied with kernel resulting in

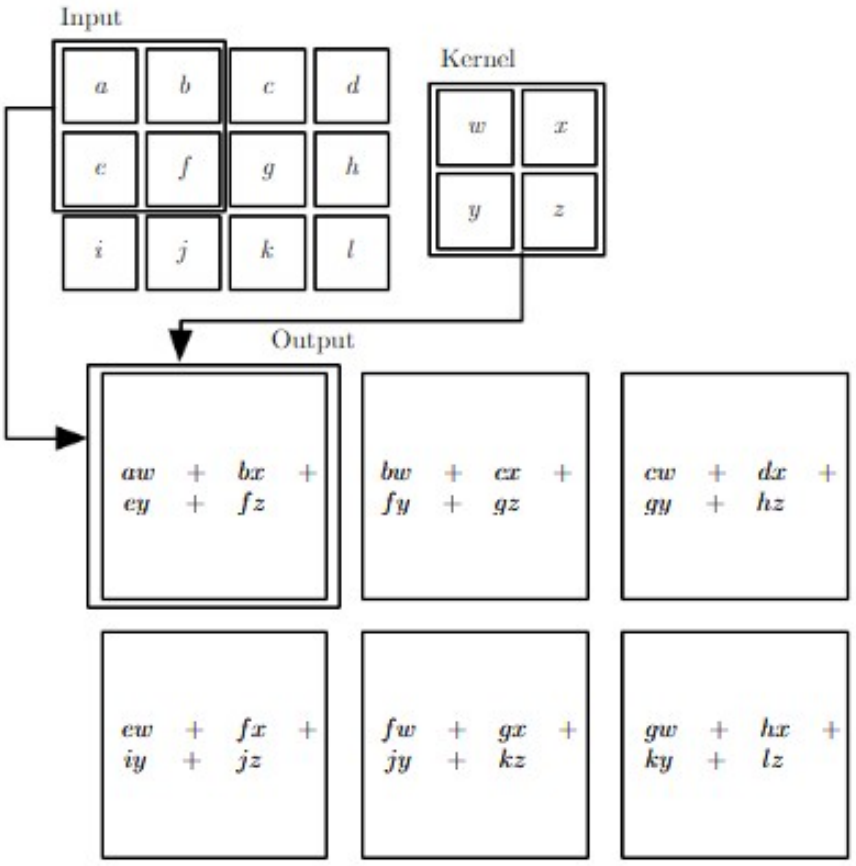


Figure 3.5: Convolution operation on input image with a kernel producing output feature map [4]

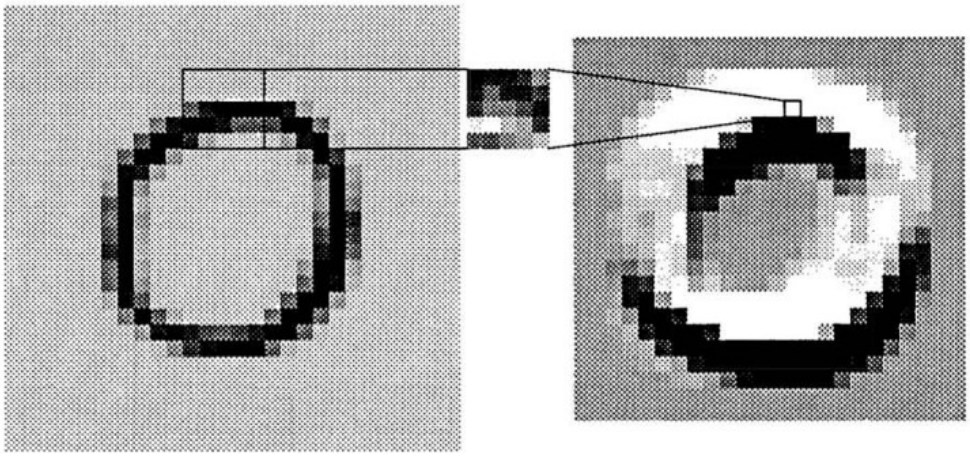


Figure 3.6: Output feature map generation from input image with weighted kernel [5]

feature map.

Figure 3.6 presents an input image essentially a color image is applied to the network model as the input. At the input data layer, which serves as an entry point of the network model performs preliminary tasks before passing through the rest of the layers of the network. At the image input layer, the pixel intensities read into multi-dimensional arrays typically 3-dimensional representing height, width, and color channel. The extracted arrays of data are then normalized using techniques such as zero center, zscore, etc. to perform on channel wise or element wise.

Figure 3.7 presents an input image to the network model which is a color image of any size (image shown is 509×360 pixels) of 3 channels. The first output from the image input layer for the given input is presented in Figure 3.8 which consists of 4 tiles where each tile represents the resized input image to the size (224×224) after initial normalizations from 3 color channels. The first tile at (1,1) represents the red channel output and tile at (1,2) represents the green channel output. The bottom left tile at (2,1) represents the blue channel output. fourth tile at (2,2) was kept blank since we considered processing only 3 color channels from the input. This can be



Figure 3.7: Input image to the network(left).

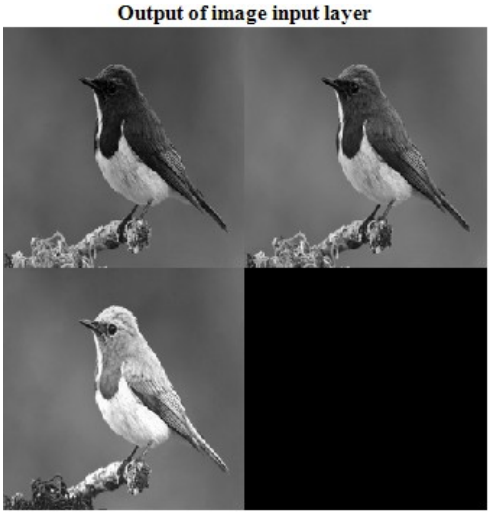


Figure 3.8: Output of the image input layer (Right)

further extended to full occupancy in case adaptation to 4 channel inputs such as CMYK used in the printing industry. For a better understanding of color channel separation, Figure 3.9 represents a color cube image applied to the image input layer. Corresponding output form layer1 is presented with label for each channel output.

The normalized input from the image input layer is then applied to the first convolutional layer. Convolution operation was performed for the input image of size (224×224) received from the image input layer with a kernel of size (11×11). The convolution operation resulted in generation of output feature map of size (55×55). Since there are 96 such filters applied to the input image, the output feature map consists of 96 output feature maps accordingly. Figure 3.8 represents the output feature map of the 1st convolution layer of the network.

The output feature map shown in Figure 3.10 consists of 100 tiles (10×10) where the last four tiles are blank since there only 96 feature maps. Since the layer conv1 which is the first convolution operation over the input image, the features extracted can be clearly seen in each output feature

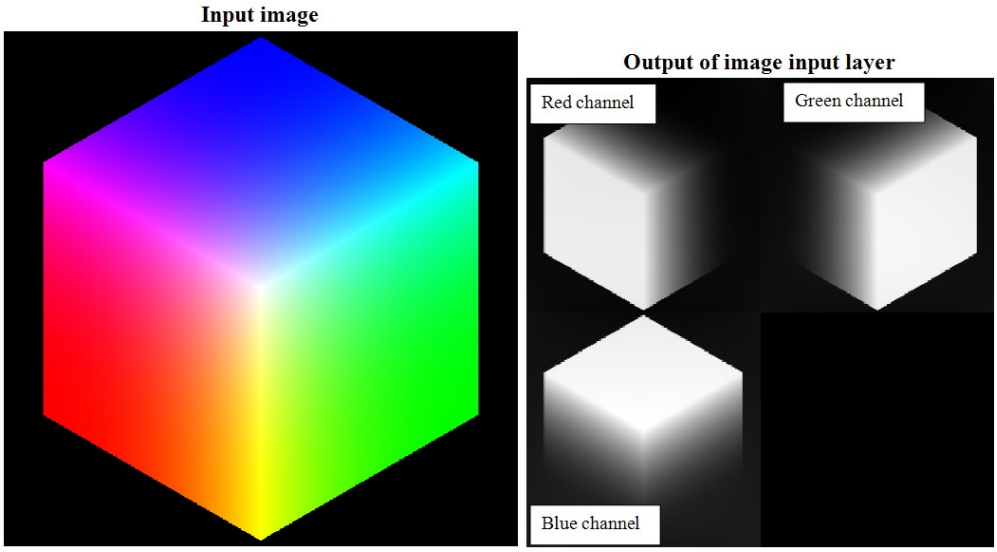


Figure 3.9: color cube input image(left), corresponding output from the image input layer(right)

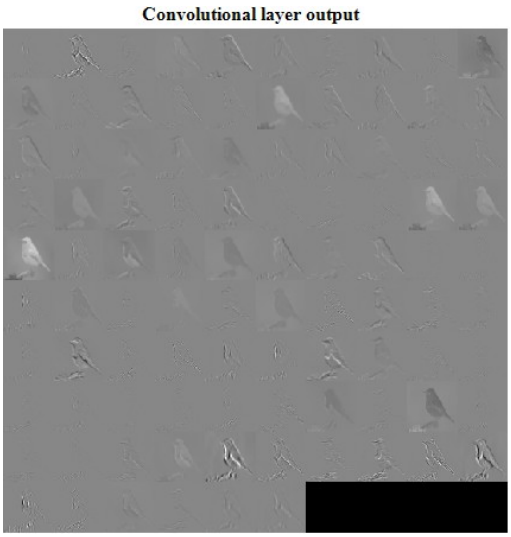


Figure 3.10: Output feature map of 1st convolution layer

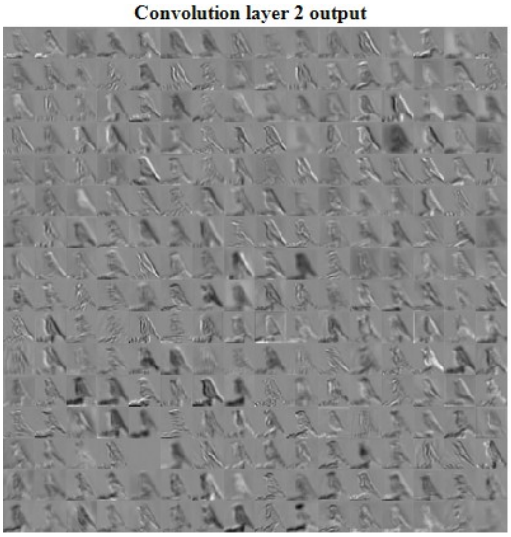


Figure 3.11: Feature map of 2nd convolution layer

map. Since conv1 layer uses 96 filter kernels to be operated over the input image, each filter is different from the other. Hence in the output feature maps, a clear distinction can be observed. The parameters used within the filter kernel are determined by the algorithm used during the training process. Therefore 96 output feature maps are different from each other. Figure 3.11 represents the output of the 2nd convolution layer.

Input to the conv2 layer is of size (55×55) which is the feature map generated from conv1 layer. The filter kernel used at conv2 layer of size (5×5) where the filter kernels were divided in to two groups of size 128 each, resulting in total of 256 filters. Those 256 filter kernels were moved across the input feature maps generated at conv1 layer finally producing 256 output feature maps. Those 256 feature maps were presented in output feature map of size (16×16) as shown in Figure 3.11.

With the fundamental definitions of convolution of two functions $w(x), g(x)$ abiding the properties of Fourier transforms can be mathematically represented in Equation 3.1.

$$(w * g)(x) = \int_{-\infty}^{\infty} w(t)\, g(x-t)\, dt \tag{3.1}$$

Since convolution is commutative which is represented with asterisk the result is written as denoted in Equation 3.2.

$$s(x) = (w * g)(x) \tag{3.2}$$

In computer convolutional neural networks, the first argument of convolution referred to as input (g), second argument is kernel (w) produces the third argument output (s) which is also referred as feature map. Since convolutional neural networks are implemented in digital computer environment, discretized inputs and outputs only considered. Therefore, discretized convolution can be denoted in Equation 3.3.

$$s(x) = (w * g)(x) = \sum_{n=-\infty}^{\infty} w(n)\, g(x-n) \tag{3.3}$$

Practically all the neural networks used in computer applications must work with more than one dimensional data. Therefore, the input for image classification based CNNs is a multidimensional array, and the kernel is multidimensional array of parameters which are adapted by the algorithm used for training of the network. The infinite summation was implemented over a finite number of array elements since the inputs and kernel parameters stored infinite number of elements held zero otherwise. Considering the input image (I)as a two dimensional array of data where each array element represents corresponding pixel value convolved with a filter kernel (k) equation 2 above can be written in general form of convolution in Equation 3.4.

$$s(p,q) = (k * I)(p,q) = \sum_{m} \sum_{n} I(p-m, q-n)\, k(m,n) \tag{3.4}$$

In practical implementation, the libraries implemented the convolution operation as cross correlation, without flip the kernel to obtain the commutative property and thus the resultant is presented in Equation 3.5.

$$s(p,q) = (k * I)(p,q) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} I(p+m, q+n)\, k(m,n) \tag{3.5}$$

For a given input size of (H_{in}, W_{in}) with input channels (C_{in}) and batch size of (N), performed convolution operation resulting output feature map of size (H_{out}, W_{out}) for output channels of (C_{out}). The output value associated with weight and bias can be represented in Equation 3.6.

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) * \text{input}(N_i, k) \tag{3.6}$$

This process can be simplified as follows. If the input image (I) performed convolution operation with filter kernel (k) by sliding the filter across the input image with specified step size known as stride (S). The output feature map (O) generated as a result which can be decomposed as elementary map of individual products from each stride constitutes the input image. It can be mathematically represented in Equation 3.7.

$$O_{i,j} = \sum_{u,v=1}^{p,q} \sum_{x,y=1}^{m,n} I_{u,v} \times K_{x,y} \tag{3.7}$$

It is a common practice in convolutional neural networks to maintain an

inputs, feature maps and output with equal size. For example, in AlexNet the most common filter kernel sizes were 3x3, 5x5, 11x11 etc. Therefore, limits of equation 2 can also be summarized to p,m as presented in Equation 3.8.

$$O_{i,j} = \sum_{u,v=1}^{p} \sum_{x,y=1}^{m} I_{u,v} \times K_{x,y} \tag{3.8}$$

Where: $O_(i,j)$ is the elementary value at position (i,j) of the output feature map. $I_(u,v)$ is the instantaneous value at position (u,v) of the input feature map. $K_(x,y)$ is the value at the position (x,y) of the filter kernel. The output feature map size (w) can be determined based on image input size (p), filter size (m), number of strides (S) and zero paddings. This can be represented in Equation 3.9.

$$w = \frac{p - m + 2 \cdot \text{padding}}{S} + 1 \tag{3.9}$$

The output value at $i, jpositionO_(i,j)$ is presented with a bias term b. The importance of addition of the bias term is to provide an offset to the output presented in Equation 3.10.

$$O_{i,j} = O_{i,j} + b \tag{3.10}$$

3.7.2 Activation function

The convolutional neural network tends to behave as a linear model since each convolutional activity introduces more linearity to its output. Since most of the CNNs consists of more than one convolutional layer in their

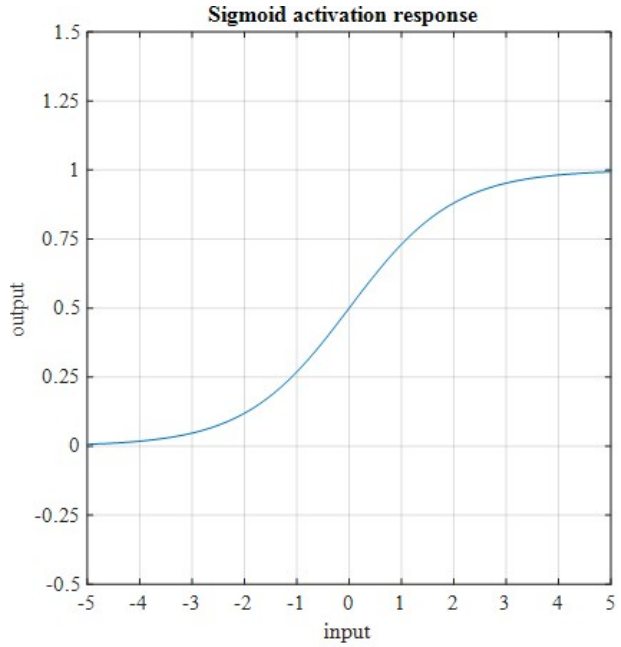


Figure 3.12: Sigmoid activation response

architecture, this introduces more linearity in their output [68] resulting in similarity to the linear regression model. This phenomenon limits the ability to learn complex patterns and relationships. Therefore, it is important to introduce nonlinearity to the network model to learn complex relationships of the input data and thereby develop a better representation of data. With the introduction of activation functions, CNNs can represent complex output features at different perspectives based on its methodology. It is also important to note that the activation functions play a vital role in maintaining the gradients during the training process of the model. However, the adaptation of a nonlinear activation function also contributes to the overall performance of the model [128] as it may introduce additional computation cost and thereby inference time. Sigmoid Activation function is represented in Equation 3.11.

$$\sigma_x = \frac{1}{1 + e^{-x}} \tag{3.11}$$

Figure 3.12 represents the activation response of a sigmoid function for a range of input values. The sigmoid activation function restricts the input values to 0 and 1 and thus it is useful for binarized classification problems. Also, it includes floating point calculations and thus more computational cost introduces. The gradient of the sigmoid function does not propagate, and it tends to zero for large number of inputs and very small number of inputs. Sigmoid activations do not involve sparse activations which leads to additional computations. Since the output of the sigmoid function is not centered, the optimization is inappropriate and the inputs tend to saturate either to 0 or 1. Hence considering these anomalies related to sigmoid functions, it was not chosen for activation function of the proposed network model. **Exponential Linear Unit (Exponential linear unit (ELU))** is another type of activation function used in CNNs for feature map activations. Mathematically described as in Equation 3.12 where all the positive inputs are kept unchanged [133]. Characteristics of ELU can be presented in Equation 3.12 for hyperparameter α where $0 < \alpha$.

$$\sigma_{\text{ELU}}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases} \tag{3.12}$$

The hyperparameter α is used as a control value for the saturation of the ELU for its negative input values. The derivative as represented in Equation 3.13 presented for positive input values-maintained unity since derivative is identity function and for the negative inputs, shifts the mean of the activation values towards zero. However, as the argument of the negative values get smaller, the tendency of ELU to become saturate could be observed with faster learning [75].

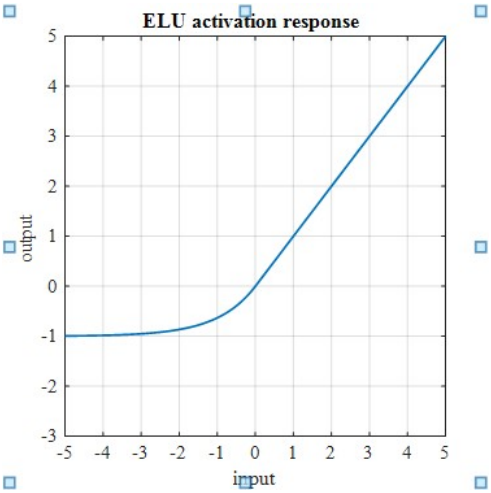


Figure 3.13: ELU activation responses for $\alpha = 1$

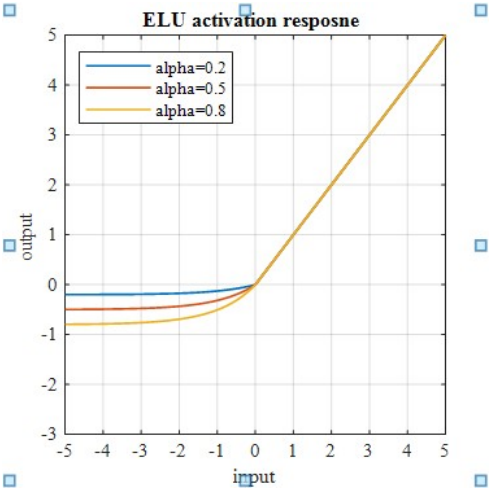


Figure 3.14: ELU activation responses for variable hyperparameter values

$$\sigma'_{\text{ELU}}(x) = \begin{cases} 1, & \text{if } x > 0 \\ \sigma_{\text{ELU}}(x), & \text{if } x \leq 0 \end{cases} \tag{3.13}$$

Figure 3.13 depicts the response of ELU activation function for range of input values. For positive integer input values, the output follows the input unchanged and for the negative inputs the output is product of hyperparameter with exponential component, which controls the values for which the ELU saturate. Figure 3.14 depicts the ELU activation response for range of inputs with different hyperparameter values where $0 < \alpha < 1$.

Rectified and Linear Unit (ReLU) is one of the most popular activation functions used in neural network models. Introduced in 1969 by Kunihiko Fukushima [134] for visual feature extraction in hierarchical neural networks, ReLU has been adapted for many applications and has still secured its popularity due to the simple computation and other benefits such as sparsity. Mathematically representing in 3.14, the output of ReLU activation follows the input unchanged for positive values of input whereas zero for negative values.

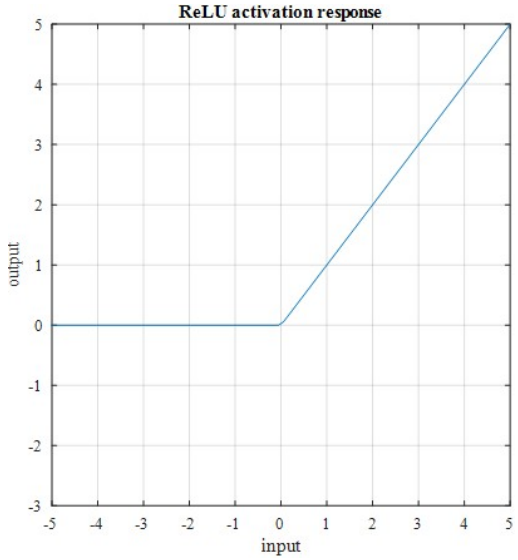


Figure 3.15: ReLU activation function response

$$\sigma_{\mathrm{ReLU}}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{3.14}$$

ReLU activation function output responses for various input values ranging from negative to positive values are presented on Figure 3.15 From the graph, for positive input values the output follows the input and zero output for the negative input values.

Since the derivative is one and non-contractive the ReLU addresses the vanishing gradient problem for the positive inputs. For the negative inputs the gradient is zero which makes all the inputs null. Since the activations become zero for negative inputs which indirectly making the system to operate faster which also leads to another problem known as dead neuron [135] where the neuron might completely inactivate due to the negative values of the input. However, considering the advantages such as sparsity suppression of vanishing gradient problem, efficient computation etc. ReLU was adapted for the proposed network model for output feature map activation as it further contributes for faster inference due to the absence of exponent

computation for negative input values of the counter activation function ELU.

3.7.3 Normalization of the intermediate results

Training a convolutional neural network with many different layers which are meant for various purposes is a tremendous task as they are susceptible to the changes of weights and configurations. Since the learning process is conducted in batches, as the inputs are passed down through the network layers, weights and parameters are updated every mini batch training session [136]. This results in slowdown the training process and extremely difficult to train network models a phenomenon known as internal covariate shift [136] [137]. Therefore, normalization techniques are used to standardize the activations within the layer and thereby stability is improved which allusively enhances the efficiency of the training process. There are different normalization techniques used in CNNs such as batch normalization, layer normalization, instance normalization, group normalization etc.

Batch Normalization

One of the most widely used normalization techniques, The Batch Normalization, standardizes the inputs to a layer for every mini batch. Normalization is performed by subtracting mean from each input and dividing by standard deviation. Considering an input I added with bias b resulting in composite input $x = I + b$ which constitutes training set where $\chi = x_(1 \dots m)$ is asset of values of x. The average of the input activations given by Equation 3.15.

$$E[x] = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{3.15}$$

Normalizing input features independently for mean of zero and unity variance, for an input feature map of d- dimension presented in Equation 3.16.

$$\hat{x}^{(d)} = \frac{x^{(d)} - E[x^{(d)}]}{\sqrt{\text{Var}[x^{(d)}]}} \tag{3.16}$$

To scale and shift the normalized value, parameters to be learned γ, β were added for each activation of the input feature to overcome the representational constraints and thus output can be denoted in Equation 3.17.

$$y^{(d)} = \gamma^{(d)}\hat{x}^{(d)} + \beta^{(d)} \tag{3.17}$$

Finally batch normalization of input feature map over mini batches with learnable parameters γ, β can be represented in Equation 3.18.

$$y^{(d)} = \gamma^{(d)}\frac{x^{(d)} - E[x^{(d)}]}{\sqrt{\text{Var}[x^{(d)}]}} + \beta^{(d)} \tag{3.18}$$

The momentum refers to the update mechanism used in batch normalization during the training process of the network model. Batch normalization layer keeps working on estimates of the calculated mean and variance values during the training. The values change over time for every mini batch and hence running estimates were kept in a specified value determined by momentum with a default value of 0.1. Mathematically the momentum can be described in Equation 3.19.

$$\hat{x}_{\text{new}} = \text{momentum} \times x_t + (1 - \text{momentum}) \times \hat{x}_{\text{est}} \tag{3.19}$$

Where, $\hat{x}_{\text{new}}$ is the running statistic,

$\hat{x}_{\text{est}}$ is the estimated statistic,

x_t is the observed value.

Modifying equation for the mean (μ) and variance (σ) the momentum updates can be derived presenting in Equation 3.20 and 3.21 respectively.

$$\hat{\mu}_{\text{new}} = \text{momentum} \times \mu_t + (1 - \text{momentum}) \times \hat{\mu}_{\text{est}} \tag{3.20}$$

$$\hat{\sigma}_{\text{new}} = \text{momentum} \times \sigma_t + (1 - \text{momentum}) \times \hat{\sigma}_{\text{est}} \tag{3.21}$$

Where, μ_{new}, σ_{new} is the running momentum update for mean and varience,

$\hat{\mu}_{\text{est}}, \hat{\sigma}_{\text{est}}$ is the estimated mean and variance.

μ_t, σ_t is observed values from previous mean and variance.

3.7.4 Pooling layers

For a given input image to a convolution neural network, features of the input are recapitulated by the convolutional layers. Depending on the filter kernel sizes, strides and zero padding the output of the generated feature map may vary. In deep neural networks where more convolutional layers are present, as the information is passed through the convolutional layers, the original features altered and thus affected the final classification. To address this issue, the feature map generated by the convolutional layer is down sampled prior to application of the next convolutional layer which is also known as local translation invariance. The pooling layer is placed after the feature activation layer from the convolutional operation to reduce the spatial dimensions of the input while retaining the original information. The pooling layer consists of a moving kernel which is moved across the

input feature map for the generation of new feature map without losing the original data. The kernel size is essentially less than the input size and moved it specific steps known as strides. The stride and the kernel size determine the spatial dimension reduction for example input feature map of size (4x4) is applied with a pooling kernel of size (2x2) with stride of 2 results in output feature map of size (2x2) which is spatial reduction by factor of 4. However, for the same feature map applied with same pooling kernel size with stride of 1 resulting output feature map of size (3x3) which is spatial reduction by factor of 1.7. Common pooling schemes are Average pooling, Max pooling, and Global average pooling. The Max pooling produces the maximum value of each pooling region from the input feature map. It results in an output feature map which represents the most activated feature within the pooling region while down sampling the input feature map. The average pooling calculates the average of the input feature map under the sliding kernel and produces output feature map respectively. Since the average is calculated rather maximum, the Average pooling offers fair representation of activated features while spatial reduction is achieved. Due to the extra calculation associated with the average pooling method which leads to additional inference time, for the proposed network Max pooling has been adapted as the pooling layer.

Mathematically the max pooling method can be explained in Equation 3.22.

$$P_{i,j} = \max_{m=0,\ldots,k_{h-1}} \max_{n=0,\ldots,k_{w-1}} I_{(i\times S+m),(j\times S+n)} \tag{3.22}$$

Where, $P_{i,j}$ is the value at position (i, j) of the output feature map,

k_h, k_w are the pooling kernel sizes,

$I_{(i\times S+m),(j\times S+n)}$ is the value at the input feature map at position $(i \times S +$

$m), (j \times S + n)$.

3.7.5 Dropout layer

Dropout is a regularization technique commonly used in convolutional neural networks. Using dropout technique, randomly selected neurons were dropped off by setting them to zero and thus avoiding them to further populating during the network training process. This technique has successfully prevented the network model from overfitting [6]. Since some of the neurons were forcefully set to zero, the links from those neurons were removed from the forward pass while no weight updates were applied on those neurons on its backward pass. Since some neurons were dropped off randomly, the remaining neighboring neurons take the representation workload for the final generation of classification in absence of dropped neurons. This results in multiple internal representations. If the neurons were not dropped, during the training process of the network the weight of the neurons settle on specific features causing a degree of specialization at some features which is known as complex co-adaptation [6]. Therefore, dropping of the neurons results in less sensitive to specific weights which causes a better generalization and thus reduced overfitting.

Figure 3.16 illustrates the process of dropout layer where Figure 3.16 a). represents the neurons linked with each other in the hidden layer of the network model. Figure 3.16 b). represents the introduction of the dropout by which certain neurons are terminated leaving some links to be eliminated. The net effect of dropout can be practically presented as shown in Figure 3.17 below.

Output feature map shown in Figure 3.17 a) represents the features learned

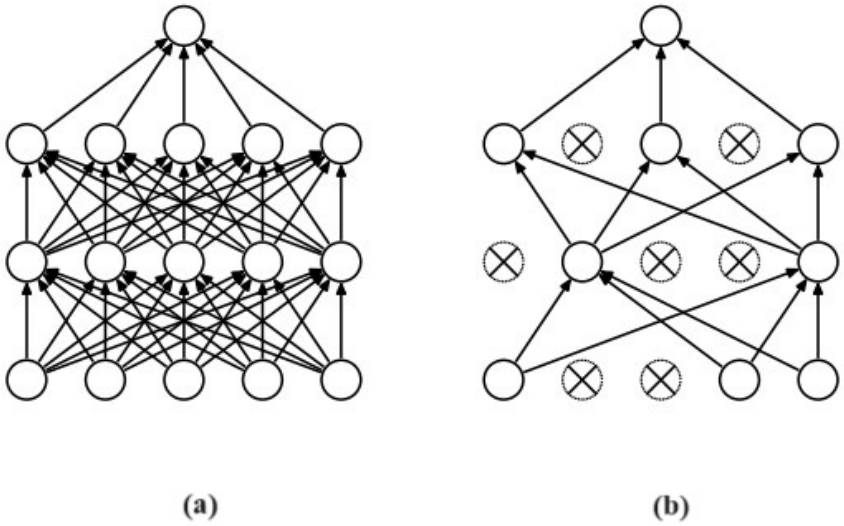


Figure 3.16: a). standard neural network b). after application of dropout [6]

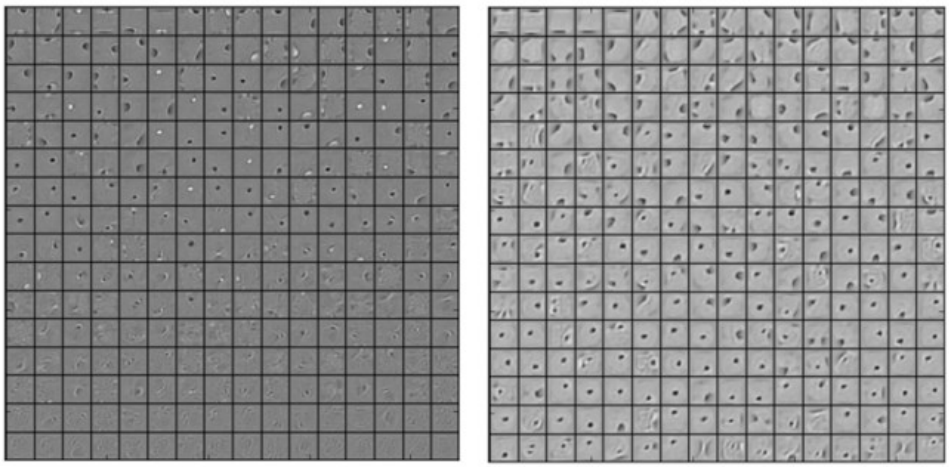


Figure 3.17: output feature map a) Without dropout. b). with dropout p=0.5 (right) [6]

from MNIST dataset with one hidden layer. The features learned by the autoencoder represented on layer of 256 rectified linear units without application of dropout using 16×16 tiles as shown in the figure. Figure 3.17 b) represents the feature map learned by the auto encoder with dropout probability of p=0.5, which is identical to the autoencoder used without dropout feature in the hidden layer. Comparing the feature maps with and without of the dropout, it can be clearly stated that features learned in Figure 3.17 a) is less meaningful compared to Figure 3.17 b) which highlights the edges detected and spots them in different tiles of the output feature map image. The feature map without dropout of which the hidden layer where neurons are connected to each other, even though expected to produce significant features, has not detected explicit features. This can be seen on the bottom part of the feature map shown in Figure 3.17 a). where the co-adaptation does not extract the important features. The term co-adaptation refers to the phenomenon where the network is trained with multiple epochs iteratively, the strong connections learn more whereas the weak connections learned less or ignored. This results in some connections of neurons offering more predictive capabilities compared to others. Therefore, the importance of dropout layer in CNN model was highlighted with a practical example. As mentioned earlier, during the training of the neural network model with large dataset the model is more susceptible to overfitting which causes low error rate on training data and more error on test data. Therefore, regularization is essential to achieve a balanced performance on both training and test data. In convolutional neural networks, better learning can be achieved by finetuning weight parameters to minimize the loss. Figure 3.18 represents neural network node with and without dropouts.

Figure 3.18 a) represents vector of outputs from layer l is transformed into

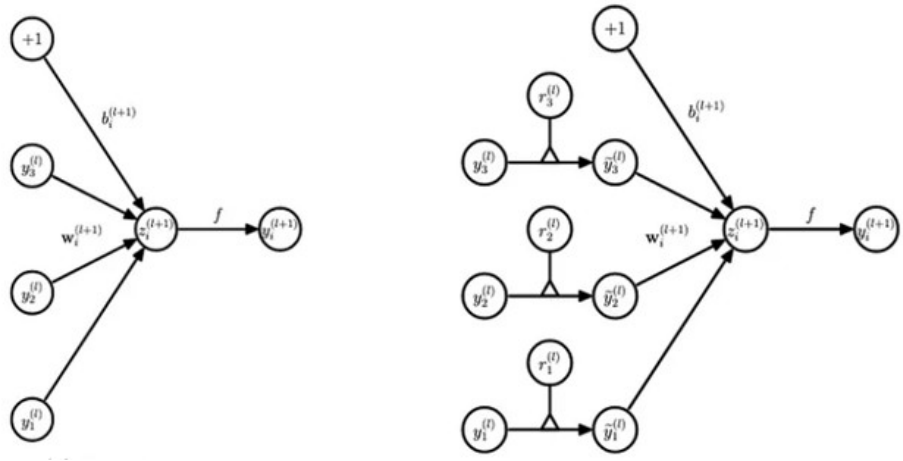


Figure 3.18: inter connection of nodes a). standard network b). dropout network [6]

vector of output of layer $l+1$ with an activation function, z during the forward propagation. Figure 3.18 b) represents the application of dropout to terminate the propagation of connections. During the forward propagation, before application of the activation function, z the inputs at the layer l are multiplied with Bernoulli variables denoted by r in element wise. The random variable, r has probability of occurrence p. The resulting outputs $\tilde{y}^{(l)}$ are considered as thinned outputs which can be used as inputs in the next hidden layer. Considering a network model with (N) hidden layers, where $z^{(n)}$ be the input vectors into the layer n, the output vector from layer n denoted by $y^{(n)}$ based on y^0 being the input x. The weights and biases at layer$n$ are denoted by w^n, b^n respectively. During the feed forward operation without dropout can be expressed as in Equation 3.23.

$$z_i^{(n+1)} = w_i^{(n+1)} y^n + b_i^{(n+1)} \tag{3.23}$$

Resulting the output feature denoted in Equation 3.24.

$$y_i^{(n+1)} = f(z_i^{(n+1)}) \tag{3.24}$$

Where function f is an activation function. During the feed forward operation with dropout, the independent variable multiplied with the input vector can be expressed with probability p as presented in Equation 3.25.

$$r_j^{(n)} \sim \text{Bernoulli}(p) \tag{3.25}$$

Thus, the output vector can be denoted in Equation 3.26as elementwise product.

$$\tilde{y}^{(n)} = r^{(n)} \cdot y^{(n)} \tag{3.26}$$

Where $*$ denotes the elementwise product. Therefore, the output at layer $(n+1)$ can be written in Equation 3.27 and output after the activation function is denoted in Equation 3.28 respectively.

$$z_i^{(n+1)} = w_i^{(n+1)} \tilde{y}^{(l)} + b_i^{(n+1)} \tag{3.27}$$

$$y_i^{(n+1)} = f(z_i^{(n+1)}) \tag{3.28}$$

From the derivation it was observed that the neuron r at any layer n can be propagated with input 1 with a probability of p or can be turned off with input 0 with probability of $1-p$. In order to further explore the possible values of the probability p the loss functions associated with the CNN model is investigated. Referring to the common loss function Mean Squared Error (Mean squared error (MSE)) [138], the error function without dropout can be presented in Equation 3.29.

$$E_{\text{ens}} = \frac{1}{2}\left(t - \sum_{i=1}^{n} w_i' I_i\right)^2 \tag{3.29}$$

With dropout mean squared error is denoted in Equation 3.30

$$E_{\text{drop}} = \frac{1}{2}\left(t - \sum_{i=1}^{n} \delta_i w_i I_i\right)^2 \tag{3.30}$$

Where, δ is the dropout rate, $\delta \sim \text{Bernoulli}(p)$

Considering the gradient decent approach, taking the gradients for both error functions can be represented in Equation 3.31.

$$\frac{\partial E_{\text{drop}}}{\partial w_i} = -t\delta_i I_i + w_i \delta_i^2 I_i^2 + \sum_{j=1}^{n} w_j \delta_j \delta_i I_i I_j \tag{3.31}$$

Let $w^{'} = p * w$ in equation and taking the partial derivative the error can be denoted in Equation 3.32 and the gradient in Equation 3.33 respectively.

$$E_{\text{ens}} = \frac{1}{2}\left(t - \sum_{i=1}^{n} p_i w_i I_i\right)^2 \tag{3.32}$$

$$\frac{\partial E_{\text{ens}}}{\partial w_i} = -t p_i I_i + w_i p_i^2 I_i^2 + \sum_{j=1}^{n} w_j p_i p_j I_i I_j \tag{3.33}$$

Considering the expectation for the network with dropout represented by Equation 3.34 and 3.35 simplified to expectation denoted in Equation 3.36.

$$E\left[\frac{\partial E_{\text{drop}}}{\partial w_i}\right] = -tp_iI_i + w_ip_i^2I_i^2 + \sum_{j=1}^{n} w_ip_jp_iI_jI_i + w_i\text{Var}(\delta_i)I_i^2 \tag{3.34}$$

$$E\left[\frac{\partial E_{\text{drop}}}{\partial w_i}\right] = \frac{\partial E_{\text{ens}}}{\partial w_i} + w_i\text{Var}(\delta_i)I_i^2 \tag{3.35}$$

$$E\left[\frac{\partial E_{\text{drop}}}{\partial w_i}\right] = \frac{\partial E_{\text{ens}}}{\partial w_i} + w_ip_i(1-p_i)I_i^2 \tag{3.36}$$

Therefore, the expectation of the gradient of network with dropout is same as the regularized network denoted in Equation 3.37.

$$E_{\text{Regular}} = \frac{1}{2}\left(t - \sum_{i=1}^{n} p_iw_iI_i\right)^2 + \sum_{i=1}^{n} p_i(1-p_i)w_i^2I_i^2 \tag{3.37}$$

Carefully examined the regularization parameter $p(1-p)$, which is maximum for $p = 0.5$ and minimum for other values. However, dropout can be changed for different regularization parameters avoiding model saturation. For the proposed cascaded network, the regularization parameter was maintained at its maximum value of 0.5.

3.7.6 Fully connected layer

The whole process of a convolutional neural network can be assumed to two stages as feature extraction and classification. Convolutional layers with their sparse connectivity extract the features of given image input while the extracted features should then be utilized for classification purposes. The objective of the fully connected layer is to provide the classification through

the trained features which are flattened into specific classes. The operation performed in a fully connected layer is connecting the neurons into specific neurons and thus the name of the layer. Within the fully connected layers sparse connections which were densely populated and restricted to limited number of neurons according to the number of output classes. Since the fully connected layer involves weights and biases which are learned during the training process it is also known as a learnable layer or a layer with learnable parameters. Multi-dimensional feature maps generated in earlier stages such as convolutional layers are flattened into single dimensional arrays during one or more fully connected layers. Generally, within the fully connected layer, neurons are connected to neurons in the previous layer. There are two operations performed in fully connected layer, linear transformation, and nonlinear activation. The linear transformation involves calculation of weighted sum of inputs with bias term. The weighted neurons were then activated by using an appropriate activation function. Limitation of the number of neurons propagated towards the final layer, dropouts are also involved. Mathematically, the linear transformation of the input vector [140] can be represented in Equation 3.38.

$$z_j = \sum_{i=0}^{m} x_i w_j + b_j \tag{3.38}$$

Where, x input vector of size $(1 \times n)$ W is the weight matrix of size $(m \times n)$ Bias vector b of size$(m \times 1)$ z is the output vector of size$(m \times 1)$ m, n is the number of neurons in present and previous layers respectively.

For the given input vector x of size $(1 \times n)$ is multiplied with the weight matrix w of size $(m \times n)$ which represents the learned parameters of the network followed by addition of the bias vector b. therefore the output of the fully connected layer is flattened into 1 dimensional array of size

specified by the weight matrix. The weighted input is then applied to activation function to introduce nonlinearity to the output. The activation function f is then applied to the output z elementally. The activation function can be chosen from various options such as tanh, ReLU, sigmoid, log sigmoid, exponential linear unit (ELU), Leaky ReLU etc. However, for the proposed network ReLU was adapted as the activation function since it operates element wise and no additional calculations needed such as exponents. The output vector y is subjected to activation function f from the fully connected layer is then represented in Equation 3.39 and 3.40 respectively.

$$y_j = f(z_j) \tag{3.39}$$

$$y_j = f\left(\sum_{i=0}^{m} x_i w_j + b_j\right) \tag{3.40}$$

Where, y is the output vector of size $(m \times 1)$ after application of the activation function. It is important to note that SoftMax activation is preferred towards the end of the fully connected layers. However, if more than one fully connected layer is involved, ReLU activations are used after the early stages of fully connecting layers.

3.7.7 SoftMax layer

SoftMax function is primarily used in convolutional neural networks as an output activation function towards the last layers for which the tasks involved multi-class classification problem. SoftMax represents the probability distribution over different classes. Since the probability distribution

of SoftMax is over discrete variables rather binary variables in sigmoid function, it can also be considered as a generalization of sigmoid function to represent discrete variables. As mentioned earlier, there are different activation functions available for classification prediction. Classification involves prediction of a class label for a given input. Different probability distributions are used depending on classification problems such as binomial probability distribution for binary classification and multinomial probability distribution for multi-class classification. Linear activation function for predicting class membership is one of the approaches where the activation function is sum of weighted input to the node of output array. Which doesn't perform additional transformations. The likelihood is therefore between 0 and 1. Since there's no transformation is performed; linear activation could result in any output value between 0 and 1 which is inappropriate for classification either binomial or multinomial problems. For this reason, the last fully connected layer is not activated using linear activation functions. Sigmoid activation is another activation function for class membership prediction which is also known as logistic function. Irrespective of the input, the sigmoid activation function output is between 0 and 1. Weighted sum of inputs with large values to be mapped onto output as 1 and the small and negative values to zero. Therefore, sigmoidal function is best suitable for binary classifications where the output is interpreted as a binomial distribution. Considering output vector of k real numbers which represents output classes, the SoftMax function normalize into probability distribution consisting of k probabilities proportional to the exponents of the inputs. Softmax function applied to the input vector z is represented in Equation 3.41.

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{k} e^{z_i}} \tag{3.41}$$

Where, z_j is the raw score to the class j, k is the total number of classes. Softmax function exponentiate each raw score of the input vector from the fully connected layer and divides by the sum of exponentiated raw scores for all classes. Normalization performed ensures that the resulting probabilities sum up to one. Since it preserves the rank order of its input values while selecting the maximum value from the generalization it is referred to as SoftMax [141].

3.7.8 Depth concatenation layer

The depth concatenation layer has been adapted to the main network of the proposed cascaded network model to enhance the generalization capabilities. Referring to the main network architecture, the inception module has been adapted towards the end of the convolutional layers. The multiple branches performed convolution operation with different kernel sizes concurrently and thus multiple activation layers have been placed to activate the neurons which are connected to the dense layer. The importance of mixed pooling has been highlighted in [195] where multi branch convolution operation is performed. However, the most important fact is to extract the feature maps generated by the convolutional layers of the branched network since the feature map is proportionate to the representational capacity of a CNN. The depth concatenation layer used in CNNs to combine those feature maps generated by each branch after the convolutional operation along the depth dimension. Depth concatenation can only be applied when a multi-branch network is present within the architecture. The objective of application of the depth concatenation layer in any CNN is to merge feature maps form different branches into a single output vector. Depth concatenation layer performs concatenation operation along the depth dimension

without subjecting to any transformation. The feature maps are stacked one after the other and this operation is used to extract information from multiple levels of abstraction in a network model. Mathematically if the feature maps are represented by $F_1, F_2, F_3, \ldots, F_n$ from different layers of the network where each feature map F_i has dimensions height (H), width (W), input channels (C_in) the depth concatenation layer produces a single output tensor with same dimensions (H, W, C_in) The depth concatenation layer is therefore an efficient way to combine feature maps from multi-branch network. It allows the network to leverage the information from multiple abstraction levels and improves the representational ability of the network model by capturing multiple information. Depth concatenation has been successfully adapted in identity shortcut connection of ResNet [8] by combining feature maps from different layers. Depth concatenation in DenseNet [196] where concatenation performed on feature maps from all previous layers facilitating reuse of features and feature propagation. Dual Path Networks [197] the depth concatenation performed on multi paths to concatenate features which facilitate information flow and enhanced representation abilities.

3.8 Methodology

The main intention of this research is to address one of the practical issues related to convolutional neural networks, the execution time for the model to develop a classification with high confidence. The exploration began with a deep dive into theoretical explanations of the neural networks and what operations are performed in the neural network model and which layers of the network account for the complex calculations. The explanation began with convolutional layer in which most of the complex computations are

performed.

3.8.1 Computations involved in Convolutional layer

Essentially the input data fed from the input layer, after subjected to pre-processing should be passed through a convolutional layer of given specifications. The feature map generated will be used by the subsequent layers. The feature map generated in the input layer will be processed within the sparse connectivity which consists convolutional operations, max pooling, batch processing and activations. In the convolutional layer, convolutional operation will be performed over the input feature map. The input image will be converted into feature map by performing the convolution with given filter with its coefficient values. Mathematically this operation can be resolved into basic operations such as complex multiplication and addition. In general, the filter coefficients are multiplied with the pixel values underneath and added together. This generates one feature for the given input image. The kernel will then be shifted by the specified stride value over the input feature map and perform the operation again and produce another single feature. Similarly, the kernel will be moved across the input image and thus produces the feature map of the specific size. Adapting the mathematical notations in [130], Mathematical representations can be rendered as expressed below.

For an image input of size (H_{in}, W_{in}) where input image height and width respectively, convolutional operation performed with kernel of size (F_m, F_n) producing an output feature map of (H_{out}, W_{out}) with zero padding (P) over the stride of (S) in combination of dilation of (D). The output feature map's height and width can be determined from Equation 3.42 and 3.43 respectively.

$$H_{\text{out}} = \left[\frac{H_{\text{in}} + 2 \cdot \text{padding}[0] - \text{dilation}[0] \cdot (\text{kernelsize}[0] - 1) - 1}{\text{stride}[0]} + 1 \right] \tag{3.42}$$

$$W_{\text{out}} = \left[\frac{W_{\text{in}} + 2 \cdot \text{padding}[1] - \text{dilation}[1] \cdot (\text{kernelsize}[1] - 1) - 1}{\text{stride}[1]} + 1 \right] \tag{3.43}$$

In our application, we maintained unity dilation and thus its value is set to 1. Since the kernel size is not changed throughout the process and its height and width are same while it's quantitative value may change from one layer to the other as a design specification. Therefore, the height and width of filter kernel (F_m, F_n) assumed to (F). Referring to Equation 3.42 and 3.43 derived above rearranging the parameters with the known information of input size and kernel size, assuming zero padding and unity stride the output feature map size can be presented in Equation 3.44 and 3.45 respectively.

Output feature map size = Input size - Filter kernel size + 1

$$H_{\text{out}} = H_{\text{in}} - F + 1 \tag{3.44}$$

$$W_{\text{out}} = W_{\text{in}} - F + 1 \tag{3.45}$$

Considering the Filter kernel with stride of (S) units, the generalized equation for the output feature map height and width can be derived as presented in Equation 3.46 and 3.47 respectively.

Output feature map size $= \frac{\text{Input size} - \text{Filter kernel size}}{\text{Stride}} + 1$

$$H_{\text{out}} = \frac{H_{\text{in}} - F}{S} + 1 \tag{3.46}$$

$$W_{\text{out}} = \frac{W_{\text{in}} - F}{S} + 1 \tag{3.47}$$

To preserve information at certain occasions it is needed to append zeros to the input. For such situations the output feature map with above specifications with stride of (S) and padding (P) can be presented by rearranging the previous set of equations, output height and width is denoted in Equation 3.48 and 3.49 respectively.

Output feature map size $= \frac{\text{Input size} + (2 \times \text{padding}) - \text{filter kernel size}}{\text{stride}} + 1$

$$H_{\text{out}} = \frac{H_{\text{in}} + 2P - F}{S} + 1 \tag{3.48}$$

$$W_{\text{out}} = \frac{W_{\text{in}} + 2P - F}{S} + 1 \tag{3.49}$$

3.8.2 Determination of complex computations in Convolution layer

It is necessary to understand the number of complex computations performed in the convolutional layers in the network model. The generalized representation of mathematical multiplications and additions are derived as follows. An input image of size ($w \times h$) and a kernel of size ($m \times n$) produce an output feature map of size ($W_2 \times H_2$) producing the coefficients. The

feature map generated consists of summed products of coefficients from kernel and input image as given in the Equation 3.50. Number of multiplications and additions per single output can be determined using Equation 3.51 and 3.52 respectively.

$$\sum_{i,j=1}^{w,h} \sum_{p,q=1}^{m,n} a_{i,j} \times b_{p,q} \tag{3.50}$$

$$\text{Number of multiplications per single output} = M \times N \tag{3.51}$$

$$\text{Number of additions per single output} = M \times N - 1 \tag{3.52}$$

if the output feature map size is w_2, H_2 Total number of multiplications and additions can be determined by using Equations 3.53 and 3.54.

$$\text{Total number of multiplications} = M \times N \times W_2 \times H_2 \tag{3.53}$$

$$\text{Total number of additions} = M \times (N - 1) \times W_2 \times H_2 \tag{3.54}$$

Since the input image can be in different format, such as RGB, CMYK, sRGB etc., the input channels may depend on the format. The number of computations depends on the number of channels in the image is constructed on. Thus, the number of color channels should be considered for computations involved. In general, for an input image of (P) number of color channels, total number of computations can be derived as presented in Equations 3.56 and 3.55 respectively.

$$\text{Total number of multiplications} = P \times M \times N \times W_2 \times H_2 \tag{3.55}$$

$$\text{Total number of additions} = P \times M \times (N-1) \times W_2 \times H_2 \tag{3.56}$$

Since each filter in convolutional layer produces a single feature map, if there are 'K' such filters present the total number of computations also change in accordance with the depth parameter. The computations can be represented with the depth parameter 'K' as follows. Therefore, the total number of multiplications and additions can be derived from Equation 3.58 and 3.57.

$$\text{Total number of multiplications} = P \times M \times N \times W_2 \times H_2 \times K \tag{3.57}$$

$$\text{Total number of additions} = P \times M \times (N-1) \times W_2 \times H_2 \times K \tag{3.58}$$

Therefore, the total number of computations given in the Equation 3.59.

$$\text{Total number of computations} = P \times M \times (2N-1) \times W_2 \times H_2 \times K \tag{3.59}$$

3.8.3 Importance of analysis of feature maps

It is known that in any neural network model, at a certain layer of the network within the sparse connectivity, the output is a feature map of a specific volume. From the earlier feed forward neural networks such as Lenet it had been a practice to use specific kernel sizes, strides as well as arrangement of intermediate layers and activations such as maxpooling layer, ReLU etc. Application of filters in convolutional layer were either one type of such as 1x1 convolution, 3x3 convolution, 5x5 convolution etc. The key ideology behind the successful neural network models such as GoogLeNet was application of multiple such layers within the network model rather depending on single operation at a time.

3.8.4 Computations in secondary network

The computation of learnable parameters and mathematical multiplications and additions were conducted in two sections separately for both main and sub network. In sub network there are 5 convolution layers, 3 max pooling layers and one fully connected layer. Learnable parameters are present only in convolutional layers and fully connected layers. Number of learnable parameters were calculated for each convolution layer and presented in Table 3.5 below.

Therefore, a total of 2.6M learnable parameters are present in sub network. However, it is important to determine the exact number of mathematical multiplications and additions present in the network. With the application of the derived equations above, the number of computations was calculated in each convolutional layer and the values were presented in Table 3.6 below.

Table 3.5: Learnable parameters in convolution layers of secondary network

Layer	**Weighted parameters**	**Bias**	**Total parameters**
Convolution 1	$11 \times 11 \times 3 \times 96$	$1 \times 1 \times 96$	34944
Convolution 2	$5 \times 5 \times 48 \times 2 \times 128$	$1 \times 1 \times 2 \times 128$	307456
Convolution 3	$3 \times 3 \times 384 \times 256$	$1 \times 1 \times 384$	885120
Convolution 4	$3 \times 3 \times 192 \times 2 \times 192$	$1 \times 1 \times 2 \times 192$	663936
Convolution 5	$3 \times 3 \times 192 \times 2 \times 128$	$1 \times 1 \times 2 \times 128$	663808
Fully connected	5×9216	1×5	46085
		Total	**2601349**

Table 3.6: Computations of secondary network

Layer	**Multiplications**	**Additions**
Convolution 1	$3 \times 11 \times 11 \times 55 \times 55 \times 96$	$3 \times 11 \times 10 \times 55 \times 55 \times 96$
Convolution 2	$2 \times 5 \times 5 \times 27 \times 27 \times 2 \times 128$	$2 \times 5 \times 5 \times 27 \times 27 \times 2 \times 128$
Convolution 3	$2 \times 3 \times 3 \times 7 \times 7 \times 384$	$2 \times 3 \times 3 \times 7 \times 7 \times 384$
Convolution 4	$2 \times 3 \times 3 \times 7 \times 7 \times 384$	$2 \times 3 \times 2 \times 7 \times 7 \times 384$
Convolution 5	$2 \times 3 \times 3 \times 7 \times 7 \times 256$	$2 \times 3 \times 2 \times 7 \times 7 \times 256$
Total	**115,649,568**	**105,878,208**

As explained in previous chapter, the main network has adapted different configurations of convolutions, maxpooling and activations to construct its classification output. There are 11 convolution layers, 4 maxpooling layers and a fully connected layer in the main network model. The learnable parameters are present only in convolution and fully connected layers, which are calculated and presented in Table 3.7 below.

Similarly, the mathematical computations are also calculated and presented in Table 3.8 below.

In addition to the convolutional layers there are other layers which perform mathematical operations which contribute to the overall execution time such as ReLU activations, Maxpooling, and Fully connect layers. However, these layers don't contribute to the learning process and hence are termed

Table 3.7: Learnable parameters of main network

Layer	**Weighted parameters**	**Bias**	**Total parameters**
Convolution 1	$11 \times 11 \times 3 \times 96$	$1 \times 1 \times 96$	34944
Convolution 2	$5 \times 5 \times 48 \times 2 \times 128$	$1 \times 1 \times 2 \times 128$	307456
Convolution 3	$3 \times 3 \times 256 \times 384$	$1 \times 1 \times 384$	885120
Convolution 4	$3 \times 3 \times 2 \times 192 \times 192$	$1 \times 1 \times 2 \times 192$	663936
Convolution 5	$3 \times 3 \times 2 \times 192 \times 128$	$1 \times 1 \times 2 \times 128$	442624
Convolution 6	$1 \times 1 \times 32 \times 256$	$1 \times 1 \times 32$	8224
Convolution 7	$1 \times 1 \times 64 \times 256$	$1 \times 1 \times 64$	16448
Convolution 8	$1 \times 1 \times 256 \times 96$	$1 \times 1 \times 96$	24672
Convolution 9	$3 \times 3 \times 96 \times 128$	$1 \times 1 \times 128$	110720
Convolution 10	$1 \times 1 \times 16 \times 256$	$1 \times 1 \times 16$	4112
Convolution 11	$5 \times 5 \times 16 \times 32$	$1 \times 1 \times 32$	12832
		Total	**2,511,088**

Table 3.8: Computations in main network model

Layer	**Multiplications**	**Additions**
Convolution 1	$3 \times 11 \times 11 \times 55 \times 55 \times 96$	$3 \times 11 \times 10 \times 55 \times 55 \times 96$
Convolution 2	$2 \times 5 \times 5 \times 27 \times 27 \times 2 \times 128$	$2 \times 5 \times 4 \times 27 \times 27 \times 2 \times 128$
Convolution 3	$2 \times 3 \times 3 \times 13 \times 13 \times 384$	$2 \times 3 \times 3 \times 13 \times 13 \times 384$
Convolution 4	$2 \times 3 \times 3 \times 13 \times 13 \times 2 \times 192$	$2 \times 3 \times 3 \times 13 \times 13 \times 2 \times 192$
Convolution 5	$2 \times 3 \times 3 \times 13 \times 13 \times 2 \times 128$	$2 \times 3 \times 3 \times 13 \times 13 \times 2 \times 128$
Convolution 6	$2 \times 1 \times 1 \times 6 \times 6 \times 32$	$2 \times 1 \times 1 \times 6 \times 6 \times 32$
Convolution 7	$2 \times 1 \times 1 \times 6 \times 6 \times 64$	$2 \times 1 \times 1 \times 6 \times 6 \times 64$
Convolution 8	$2 \times 1 \times 1 \times 6 \times 6 \times 96$	$2 \times 1 \times 1 \times 6 \times 6 \times 96$
Convolution 9	$2 \times 3 \times 3 \times 6 \times 6 \times 128$	$2 \times 3 \times 2 \times 6 \times 6 \times 128$
Convolution 10	$2 \times 1 \times 1 \times 6 \times 6 \times 16$	$2 \times 1 \times 1 \times 6 \times 6 \times 16$
Convolution 11	$2 \times 5 \times 5 \times 6 \times 6 \times 32$	$2 \times 5 \times 4 \times 6 \times 6 \times 32$
Total	**118,016,928**	**105,489,984**

non-learnable layers. This section focuses on the computations from those layers and their effect on total execution time.

3.8.5 Computations from the ReLU activations

A feature map generated by convolutional layer is invalid unless passed through activation function. There are different activations used in neural networks such as ReLU, Sigmoid, tanh etc. However, ReLU is mostly used activation type among most network models. ReLU compares input value from every single neuron and replaces negative values with zero and positive values unchanged. For a single neuron it requires one comparison and single multiplication to produce output to the next stage. If ReLU has (N) neurons, then total number of multiplications $= N \times$ computation for single neuron. The mathematical operations involved are comparison and thresholding. Number of comparisons equivalent to the number of neurons (N). Thresholding operations don't involve additional calculations other than setting to zero. Therefore, the total number of calculations is equivalent to the number of neurons in the layer of interest. If the feature map is generated in a particular layer of size $H \times W \times K$ (height, width, and depth of feature map) the total number of computations involved in ReLU layer is given by Equation 3.60.

$$\text{Computations in ReLU layer} = W_2 \times H_2 \times K \tag{3.60}$$

3.8.6 Computations from Maxpool layer

When the feature map is generated from the convolutional layer, with every stride of the filter over input data, the maximum value must be chosen from the feature map. Hence this process is known as maxpooling. For an input feature map of size $(W_2 \times H_2 \times P)$ where width, height, and number of channels over a pooling filter of size $(U \times V)$ width and height respec-

tively with stride of (S) and depth of (K) the number of computations is determined by the number of pooling regions and number of comparisons within each pooling region. Number of pooling regions are determined by the product of horizontal and vertical regions as presented in Equation 3.61.

$$\text{Number of regions} = \left(\frac{H_2 - U}{S} + 1\right) \times \left(\frac{W_2 - V}{S} + 1\right) \tag{3.61}$$

The number of comparisons made within the pooling region is determined by the number of elements in the pooling filter. Therefore, the number of comparisons required for pooling region is given by Equation 3.62

$$\text{Number of comparisons within the pooling region} = U \times V \tag{3.62}$$

Total number of comparisons in pooling layer is the product of number of comparisons within the pooling region and number of pooling regions. Therefore, the total number of computations involved in pooling layer can be denoted in Equation 3.63.

$$\text{Number of computations} = \left(\frac{H_2 - U}{S} + 1\right) \times \left(\frac{W_2 - V}{S} + 1\right) \times U \times V \times K \tag{3.63}$$

3.8.7 Computations from Fully connect layer

Referring to any convolutional neural network, as the input image is passed through the layers starting with input layer producing initial feature map, followed by sparse connections of convolutional layers should converge to specific neuron which ultimately represents one of the classes it is designated. Within the dense layer, each neuron is connected to every element of the input vector. Dot product is calculated between input vector and weights followed by addition of bias value. For input vector of size (J) with bias (b), the computations can be derived as the following equations. Since each neuron in the fully connected layer is connected to every element in the input, dot product requires (J)multiplications and $(J-1)$ additions. Therefore, the number of computations is approximately $(2J)$. If bias of (b) present, the total number of computations per neuron is given in Equation 3.65.

Number of multiplications per neuron $= J$

Number of additions per neuron $= J - 1$

$$\text{Number of computations per neuron} = (2J - 1) + b \tag{3.64}$$

If (I) number of neurons are present at the fully connect layer, total number of computations involved in the fully connect layer is given in equation 3.65.

$$\text{Total number of computations} = I\left((2J - 1) + b\right) \tag{3.65}$$

3.9 Implementation

The proposed system was implemented using MATLAB. The Deep Learning Toolbox of the MATLAB is a powerful tool that offers a wide range of capabilities including creation of a new network model from scratch, analytical tools, execution tools, profiling tools and deployment tools. Unlike other sophisticated software tools which require expertise in TensorFlow, Keras etc. MATLAB doesn't require specialized training for the user. In terms of user friendliness and flexibility Deep Learning toolbox offers the best option where anybody can easily configure, analyze and conduct training and verification and finally testing of a model can be completed using the same software tool whereas many other software needs special tools for configuration, training, testing and verification, GPU programming for training and execution which can be cumbersome to the user to develop a trained network model. MATLAB deep network designer together with another tool which is linked within the Deep Learning Toolbox is used to create a brand-new network or to modify an existing network model. The proposed neural network model where both main network and secondary network were solely constructed and analyzed using this software tool. The Deep Network Designer consists of a single network component which can be re-configured. Therefore, every single layer of a network can be created, and their parameters can be modified by changing the values in the properties box. An illustration of the deep network designer is presented in Figure 3.20 which shows the essential components of neural network on the left, workspace at the center and parameter edit boxes on the right.

Once the network layers are completed with required specifications, the next stage is to analyze the created network model. With the built-in analyzer tool, as shown in Figure 3.20 the network model's preliminary

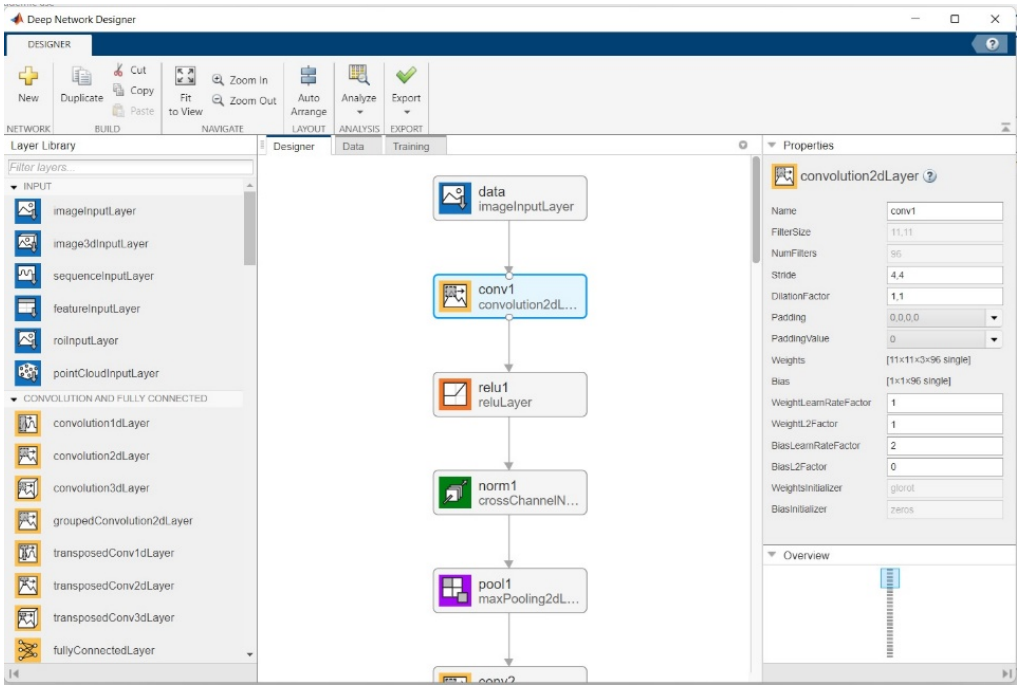


Figure 3.19: Deep network designer interface

specifications are cross checked, if any mismatch presents, an error will be shown. Kernel sizes and strides must be carefully selected after manual calculations. Arbitrary usage of filter parameters results in drop off the information to be carried on to the next stages and may violate design rules. In the case of property mismatch a warning will be issued and it must be cleared up before the training process starts.

With no errors or warnings present from the Deep Network Designer, the designed new network must be trained with pre-processed dataset. Selection of the dataset is therefore the next step to take. Navigating to the data tab in the deep network designer, the data set can be selected and certain modifications such as separating the training and validation data, scaling the data, rotation of the data as a ratio can be done. Once the training dataset is loaded the next step is to define the training parameters. These parameters as shown in Figure 3.21, must be carefully selected to make sure efficient and better training of the neural network. The hardware resources allocation and feasibility of training with available resources are completely

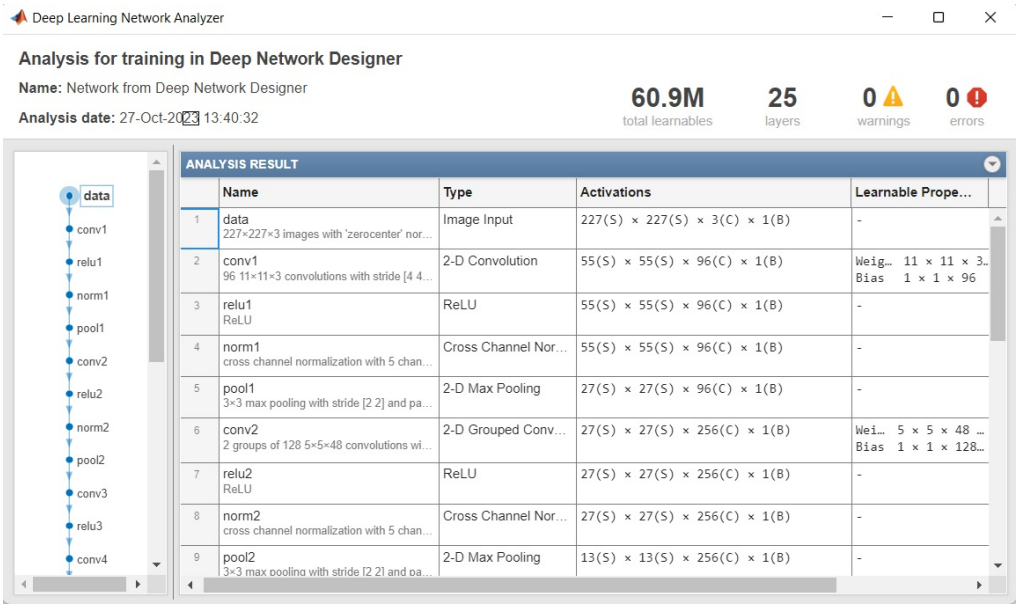


Figure 3.20: Deep Network Designer analysis report for the network model

dependent upon these parameters. Some important parameters are mentioned below. Epochs refers to the number of times the neural network is trained with the dataset. For larger datasets the epochs should be at least 30 or higher for better classification accuracy and for smaller datasets it should be kept below the default value. The validation frequency refers to how often the network is being validated during each epoch. Max epochs and validation frequency together determine how long the training process would take to complete. Mini batch size refers to the number of training samples used in each iteration. Mini batch size should be the ratio of Total number of observations to the validation frequency to ensure the whole dataset is occupied during the training of each epoch. Based on the available Graphical Processing Units and memory these options should be placed. Most important parameters of concern are initial learning rate, mini batch size, max epochs, validation frequency, execution environment etc. increase of the mini batch size reduces the training time but also requires more graphical memory which is a limiting constraint. Reduction of batch epochs and validation frequency reduces training time but also

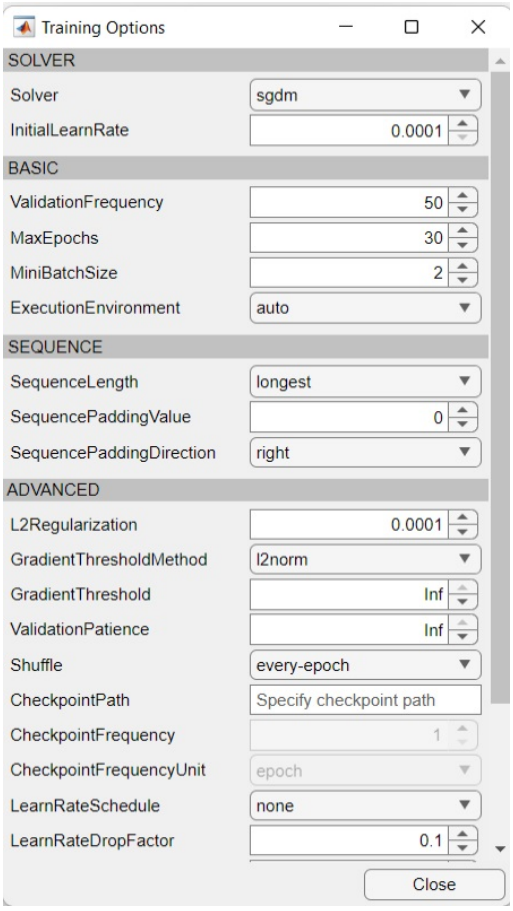


Figure 3.21: deep network designer training options

accuracy. Hence the selection of these parameters was done by trial and error.

Once the training process is successfully completed with no errors, the Deep Network Designer visualizes the training progress in a graph as shown in Figure 3.22. The progress is presented in two plots as accuracy and loss in each iteration which can further be analyzed to comment on performance of the network model.

3.9.1 MATLAB implementation and deployment

The proposed cascaded convolutional neural network (CNN) model was implemented and tested using MATLAB R2023a's Deep Learning Toolbox.

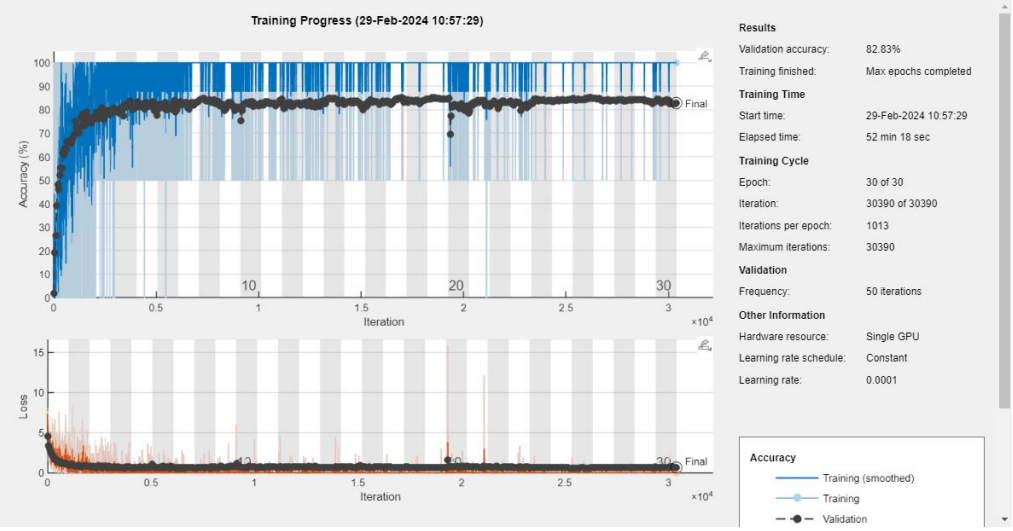


Figure 3.22: Training progress graph

This toolbox offers a comprehensive framework for designing, training, and deploying deep learning models on both CPUs and GPUs. MATLAB's integration with GPU Coder and Deep Network Designer facilitated efficient experimentation and allowed for the deployment of the lightweight CNN model on embedded platforms, such as the Raspberry Pi 4B. Development Environment and Configuration: All implementations and simulations were conducted on a workstation equipped with an Intel Core i5 processor, 16 GB of RAM, and an NVIDIA RTX 3050 GPU, with CUDA acceleration enabled. MATLAB's Parallel Computing Toolbox was used to parallelize the training processes and enhance convolutional computations. The trained model was subsequently deployed and evaluated on the Raspberry Pi platform to assess inference latency and power efficiency. Dataset Preparation and Preprocessing:

The dataset used for training and evaluation consisted of 200 object categories, representing common environmental objects relevant to service robot guidance. All images were resized to a uniform resolution of 227 x 227 x 3 pixels to meet the input requirements of the designed Convolutional Neural Network (CNN).

To enhance generalization and robustness, various data augmentation techniques were applied, including:

- Random rotation (±15°)
- Horizontal flipping
- Random cropping
- Intensity normalization

MATLAB's 'imageDataAugmenter' and 'augmentedImageDatastore' functions were utilized to dynamically augment the dataset during training. This approach helped prevent overfitting and improved the stability of convergence.

Network Architecture Implementation: The cascaded CNN model was implemented in two stages:

(a) Main CNN Network: The primary feature extraction network was constructed using the layerGraph function, which creates a sequence of convolutional layers, ReLU activations, and max-pooling layers. The architecture included:

- Conv1–Conv5: Sequential convolutional layers that extract progressively complex features.
- Conv6–Conv11: Multiple parallel convolutional branches were added using addLayers and connectLayers to perform multi-scale feature extraction.

A depth concatenation layer merged the outputs from all parallel branches, which were then passed to a fully connected layer. The first stage of the

inference pipeline was completed with a softmax layer and a classification output layer.

(b) Secondary CNN Network: The secondary CNN was designed as a lightweight refinement network consisting of five convolutional layers (snConv1–snConv5) and one fully connected layer. This network was selectively activated based on a confidence threshold determined from the output probabilities of the first stage.

Training Procedure: Both Convolutional Neural Networks (CNNs) were trained using stochastic gradient descent with momentum (SGDM) optimization. The following hyperparameters were empirically determined to achieve optimal convergence:

- Initial Learning Rate: 0.001
- Momentum: 0.9
- Mini-Batch Size: 32
- Maximum Epochs: 30

Training and validation accuracy were monitored using MATLAB's 'trainingOptions' and 'trainNetwork' functions. The secondary CNN was specifically trained on misclassified and low-confidence samples from the main network, allowing it to specialize in more complex image categories.

Confidence Threshold and Cascaded Inference: After training, a confidence threshold of 0.75 was empirically determined. During the inference process, the primary convolutional neural network (CNN) conducted the initial classification and generated a confidence score from the softmax output. If the highest probability was below 0.75, the image was automatically redirected

to a secondary CNN for reclassification. Early exits with higher classification confidence were passed onto the output interpreter for the final classification result, which informed the decision-making of subsequent systems. This cascaded process was implemented using MATLAB's classify function along with a custom conditional script to manage both networks sequentially.

Performance Evaluation and Deployment: The initial testing and evaluation of the network model were conducted on a standard computer running a Windows operating system. The trained model's performance was assessed using MATLAB's 'evaluateDetectionPrecision' function, and the test dataset was used for evaluating both network models. Additionally, live images captured from a camera connected to the computer were utilized to assess the real-time performance of the proposed cascaded model.

Metrics such as classification accuracy, inference time, and memory utilization were recorded on both the workstation and the Raspberry Pi. For each test image, the evaluation was performed five times with an interval of three seconds between tests, and the results were then averaged.

Embedded Deployment: The final trained model was converted for embedded deployment using MATLAB GPU Coder along with the MATLAB Support Package for Raspberry Pi Hardware. The workflow involved the following steps:

- Exporting the trained model using the 'exportONNXNetwork' function
- Generating optimized C/C++ code for inference using the 'codegen' function.
- Deploying the generated code onto the Raspberry Pi board, which

supports ARM NEON acceleration.

Prior to the final deployment of the network model, remote debugging was performed. This implementation enabled real-time image classification with an attached image capturing device. On the Raspberry Pi 4, both USB and CSI interfaces are available; however, testing was limited to the USB interface for a better comparison of results.

3.9.2 Testing methods

In this research, the speed of execution of the neural network, which is one of the performance parameters, is being evaluated. It is important to understand what the performance evaluation methods for a convolutional neural network are. Generally, the performance of any convolutional neural network is described in two ways classification accuracy and speed of execution. To assess the accuracy of the model, there are various testing methods such as Hold-out validation, cross validation, k fold cross validation, confusion matrix, Precision recall and $F1$ score evaluation, mean average precision map etc. Whereas the speed of execution is measured using methods such as Inference time measurement, Batch inference time measurement, profiler tools, hardware specific tools, batch size variation, model parallelism, model pruning, Latency measurement, Profiling on real data etc. However, this research focuses on the speed of execution.

3.9.3 Inference time measurement

For a given convolutional neural network model, inference time refers to the amount of time taken by the neural network to produce a classification

output when an acceptable input is fed to it. The network model is pre-trained with an accurate dataset of sufficient quantities. This quantifies how fast the neural network model can make predictions for new data which is not used during the training process. The inference time measurement is essential in time critical real-world applications such as active safety systems in automobiles, product sorting in high-speed process control etc. For applications which require low latency, it is necessary that the network model produces accurate predictions in a limited time. Hence the inference time can also be considered as a feasibility assessment for deployment of convolutional neural network in real world applications. The inference time measurement involves calculations of the execution time during a complete forward pass where the input data is passed through all the layers of the convolutional neural network until the classification output is produced. The time measurement can be achieved using various tools such as timing libraries, profiling tools etc. This entire process can be explained in five steps.

1. Select the input: The input data sample or batch of samples to be classified. It is necessary that the input samples were not used during the network model training process, and it should be real world data.

2. Start timing: record the start time before feeding the input sample to the pre-trained neural network.

3. Execute forward pass: pass the input data through the neural network's all the layers starting the image input layer up to classification output layer where the classification prediction is obtained.

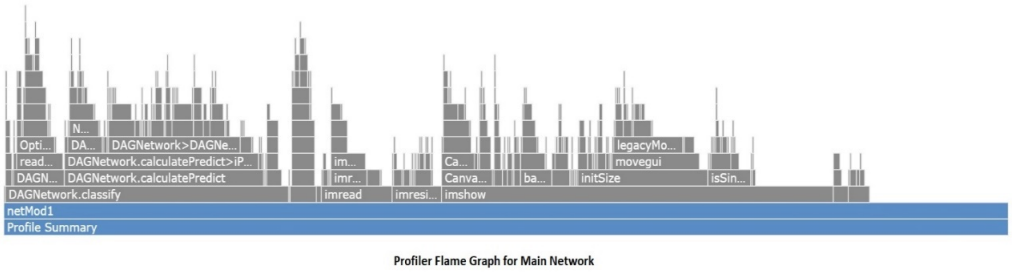
4. Stop timing: record the end time once the forward pass is completed and classification output is produced.

5. Calculate the inference time: The time elapsed or difference between stop time and start time gives the inference time.

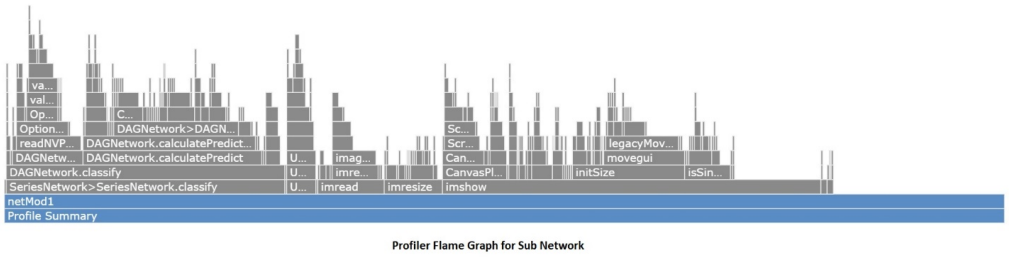
The inference time can vary depending upon hardware resources, input data size and neural network architecture. Availability of the hardware resources such as GPU rather than CPU only accelerates the computations and addition of external hardware accelerators should also reduce the execution time. A comparison is presented in the Results Comparison section in chapter 4. The number of learnable layers, essentially the dense layers, contributes significantly to the execution time. A neural network model with more dense layers thus learnable parameters takes more time to execute for a given input data. The size of the input data also plays a vital role in inference time as the input data should be pre-processed prior to applying through the neural network which includes application of algorithms such as resizing and compression. The large input data compression or resizing could also result in a loss or distortion of information which is undesired as it affects the accuracy. Hence input data size is also an important factor which decides the inference time.

3.9.4 Execution speed measurement using profiler tools

Profiler tools can be used to measure the execution performance of any convolutional neural network. Since the profiler tool can exaggerate the timing by individual layers, it is essential to identify if any bottlenecks are present in the network, any kind of inefficiencies associated with the network, or perform optimizations by observations. For this research MATLAB based profiler was used. The profiler should be configured prior to invoking it by setting up the target hardware. The profiling data is in the form of layer-wise timing, memory usage, data transfer etc. During the inference,



(a) Profiler flame graph for main network model.



(b) Profiler flame graph for secondary network model

Figure 3.23: Profiler output for a single execution of the main network and the sub network

as the profiler was invoked, it collects data from timing, memory usage and performance matrices. The profiler output for a single execution of the main network and the sub network is presented in the Figures 3.23a and 3.23b.

The flame graph is a visual representation of time taken by the MATLAB to execute each function while running the code. Each function which was run is represented by the bars in the flame graph and represented in two color schemes where blue color represents user defined functions, and the gray color represents the MATLAB functions. They are placed in hierarchical order where the parent functions are placed in lower section and child, or sub functions are placed in higher sections. The width represents the percentage of the total time.

3.9.5 Hardware testing

This research was carried out using a standard computer with GPU with specifications outlined in Table 3.9. Both the training and verification were done on the same computer. For comparison of the data the trained network was imported for different computers and data was acquired.

Table 3.9: Hardware specifications of the computer

Peripheral	**Specifications**
Random Access Memory (RAM)	16 GB
Central Processing Unit (CPU)	Core i5-9400 – 6 cores, 9MB cache, 4.10 GHz
Graphical Processing Unit 1 (GPU)	NVIDIA RTX 3060 – 12GB, 3584 CUDA cores, 1.78 GHz
Graphical Processing Unit 2 (GPU)	NVIDIA GTX 1050 Ti – 4GB, 768 CUDA cores, 1290 MHz
Graphical Processing Unit 3 (GPU)	NVIDIA GTX 1050 – 2GB, 640 CUDA cores, 1354 MHz
NVMe Solid State Drive (SSD)	250GB – 3500 MB/s, 375,000 IOPS

Hardware testing evaluates the performance of CNN model and its compatibility on hardware platforms. The assessment describes how better the network performance on specific hardware and further analysis suggests optimizations for performance enhancement. Major aspects of the hardware testing are hardware compatibility assessment and model deployment.

The proposed cascaded CNN model was tested on a hardware-based mobile robot platform. The figure 3.24 shows a fully integrated mobile service robot prototype designed for human detection and navigation in structured environments, such as airports. The system is modularly constructed with two main subsystems: a vision unit mounted vertically on a lightweight aluminum frame, and a mobile robotic platform that serves as the base for

locomotion.

Vision Unit:

The vision unit, mounted on top of the chassis, consists of the following components:

- A USB camera positioned at the top for wide-angle environmental perception.
- A Raspberry Pi 4 Model B, housed in a protective enclosure, serving as the central processing unit for real-time image acquisition and convolutional neural network (CNN) inference.
- A Raspberry Pi Camera Module v2.1, directly connected to the Raspberry Pi 4 via the dedicated Camera Serial Interface (CSI) port, enabling high-resolution, low-latency image capture.
- A 7-inch touch display mounted below the cameras, providing an interactive interface for system monitoring, manual control, and real-time visualization of detection results.

This unit is responsible for continuously capturing video streams from both cameras, preprocessing the frames, and executing a proposed deep learning model, along with comparative state-of-the-art CNN architectures, to detect and localize humans within the airport environment.

Mobile Robotic Platform:

The lower section of the platform contains the locomotion and control hardware, including:

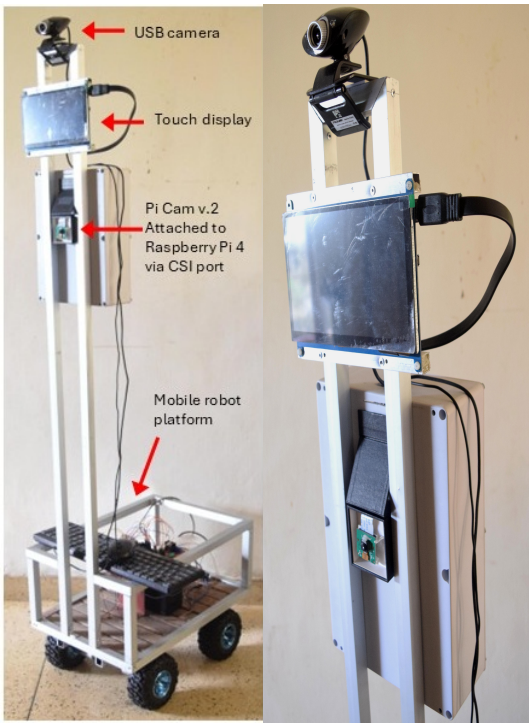
- A 4-wheel drive (4WD) differential-drive chassis, designed with rugged rubber tires to ensure stable movement across indoor surfaces.

- An Atmega2560 microcontroller that serves as the low-level motion controller. This microcontroller receives high-level navigation commands from the Raspberry Pi and converts them into precise motor PWM signals for accurate directional control.
- The entire platform is powered by a rechargeable battery pack, with wiring neatly routed through the frame for safety and maintainability.

System Integration and Functionality: The robot operates autonomously, utilizing a vision unit that processes visual input to identify human presence through the proposed CNN model. This model has been benchmarked against other state-of-the-art architectures, such as AlexNet, GoogLeNet, EfficientNet, and MobileNetV3, under the real-world lighting conditions typical of airport terminals. Once humans are detected, their coordinates are communicated to the Atmega2560 controller to initiate appropriate navigation behaviors such as approaching, avoiding, or queueing allowing the robot to serve passengers safely and efficiently.

The hardware tests were conducted in real-time under real-world conditions to evaluate the system's performance. The image shown in Figure 3.25 depicts a field test of the proposed service robot operating outdoors. During the tests, the robot encountered various real-world scenarios, including ambient daylight, moving vehicles like city buses and motorcycles, and pedestrian infrastructure. This demonstrates the robot's robustness beyond controlled indoor environments.

The vision unit is continuously active, processing visual input to detect humans and obstacles while the four-wheel-drive (4WD) platform safely maneuvers along the terrain. This testing validates the system's adaptability to dynamic and unstructured environments, such as those found at airport perimeters and transit zones.



(a) (b)

Figure 3.24: Mobile robot platform equipped with a vision unit. (a) The vision unit is mounted on the mobile robot structure. (b) Vision Unit Components: USB Camera, Touch Display Unit, Raspberry Pi 4, and Pi Camera v2.1



Figure 3.25: Real-time results from dynamic outdoor environment.

3.10 Proof of the concept

Mathematical modeling for the proposed network was presented in the previous chapter with the extensive interpretation of each layer's computations involved. Mathematical interpretation of AlexNet architecture and comparison of the computations involved should be contrasted with proposed network model in order to prove the concept. Considering the network architecture of AlexNet, the major contribution to the complex computations involved are from convolutional layers and fully connected layers apart from the other layers which contribute to the learnable parameters. For an image input to the AlexNet of size $(w \times h)$, convolutional operation performed with a kernel of size $(m \times n)$ producing an output feature map of $(W_2 \times H_2)$ with no zero padding over stride of one. By applying Equation 3.48 and 3.49 derived previously, the output feature map size for the AlexNet can be derived. The number of computations involved in convolutional layers can be derived from Equation 3.59 by adapting Equation 3.57and 3.58 for AlexNet. Since the AlexNet uses the rectification and linearization unit (ReLU) as the activation function for the output feature maps generated from the convolutional layers, which also involves in mathematical computations the number of computations involved in ReLU layer can be derived by applying Equation 3.60for the AlexNet. Similarly, the computations involved in AlexNet from the maxpool layer can also be calculated by applying Equation 3.63 derived above. AlexNet architecture consists of three fully connected layers which also contribute for significantly large learnable parameters and hence large number of computations. The computations involved in AlexNet fully connect layers can be obtained by applying the Equation 3.65.

3.10.1 Determination of computations involved in AlexNet convolutional layers

In any type of neural network architecture, the mathematical computations involved can be explained as computations in convolutional layers, computations in fully connected layers and computations involved in non-learnable layers such as ReLU, Maxpool etc. In this section, the computations taken place are described into its deepest level as number of multiplications and additions beginning with an example. An input image of size $(w \times h)$ produces an output feature map of $(W_2 \times H_2)$ with a kernel or filter of size $(m \times n)$ as shown in the Figure. The filter will be moved across the input pixels with given steps known as strides denoted by (S). Based on the requirement, sometimes zeroes will be appended to the output feature map to comply with the feature map size constraints. By applying this generalized equation 3.57, 3.58 and 3.59 to the convolutional layers on transfer learned AlexNet and proposed cascaded network the numerical results were obtained and presented in following tables. The total number of computations involved in transfer learned AlexNet is presented in Table 3.10.

Table 3.10: computations involved in convolutional layers of Alex net.

Layer	**Multiplications**	**Additions**	**Total Computations (Expr.)**	**Total**
Conv 1	$11^2 \times 55^2 \times 3 \times 96$	$(11^2 - 1) \times 55^2 \times 3 \times 96$	$(2 \times 11^2 - 1) \times 55^2 \times 3 \times 96$	209,959,200
Conv 2	$5^2 \times 27^2 \times 48 \times 256$	$(5^2 - 1) \times 27^2 \times 48 \times 256$	$(2 \times 5^2 - 1) \times 27^2 \times 48 \times 256$	438,939,648
Conv 3	$3^2 \times 13^2 \times 256 \times 384$	$(3^2 - 1) \times 13^2 \times 256 \times 384$	$(2 \times 3^2 - 1) \times 13^2 \times 256 \times 384$	282,427,392
Conv 4	$3^2 \times 13^2 \times 192 \times 384$	$(3^2 - 1) \times 13^2 \times 192 \times 384$	$(2 \times 3^2 - 1) \times 13^2 \times 192 \times 384$	211,820,544
Conv 5	$3^2 \times 13^2 \times 192 \times 256$	$(3^2 - 1) \times 13^2 \times 192 \times 256$	$(2 \times 3^2 - 1) \times 13^2 \times 192 \times 256$	141,213,696
Total	**651,767,328**	**618,575,616**		**1,270,342,944**

Similarly, by applying the equation 3.57, 3.58 and 3.59 to the cascaded network the corresponding multiplications, additions and computations were calculated and tabulated. Computations conducted separately for main network and sub network results are presented in Table 3.11 and 3.12 respectively.

Table 3.11: computations involved in convolutional layers of main network

Layer	**Multiplications**	**Additions**	**Total Computations (Expr.)**	**Total**
Conv 1	$11^2 \cdot 55^2 \cdot 3 \cdot 96$	$(11^2 - 1) \cdot 55^2 \cdot 3 \cdot 96$	$(2 \cdot 11^2 - 1) \cdot 55^2 \cdot 3 \cdot 96$	209,959,200
Conv 2	$5^2 \cdot 27^2 \cdot 48 \cdot 256$	$(5^2 - 1) \cdot 27^2 \cdot 48 \cdot 256$	$(2 \cdot 5^2 - 1) \cdot 27^2 \cdot 48 \cdot 256$	438,939,648
Conv 3	$3^2 \cdot 13^2 \cdot 256 \cdot 384$	$(3^2 - 1) \cdot 13^2 \cdot 256 \cdot 384$	$(2 \cdot 3^2 - 1) \cdot 13^2 \cdot 256 \cdot 384$	282,427,392
Conv 4	$3^2 \cdot 13^2 \cdot 192 \cdot 384$	$(3^2 - 1) \cdot 13^2 \cdot 192 \cdot 384$	$(2 \cdot 3^2 - 1) \cdot 13^2 \cdot 192 \cdot 384$	211,820,544
Conv 5	$3^2 \cdot 13^2 \cdot 192 \cdot 256$	$(3^2 - 1) \cdot 13^2 \cdot 192 \cdot 256$	$(2 \cdot 3^2 - 1) \cdot 13^2 \cdot 192 \cdot 256$	141,213,696
Conv 6	$1 \cdot 1 \cdot 6^2 \cdot 256 \cdot 96$	$1 \cdot 6^2 \cdot 256 \cdot 96$	$2 \cdot 1 \cdot 6^2 \cdot 256 \cdot 96$	1,769,472
Conv 7	$1 \cdot 1 \cdot 6^2 \cdot 256 \cdot 64$	$1 \cdot 6^2 \cdot 256 \cdot 64$	$2 \cdot 1 \cdot 6^2 \cdot 256 \cdot 64$	1,179,648
Conv 8	$1 \cdot 1 \cdot 6^2 \cdot 256 \cdot 32$	$1 \cdot 6^2 \cdot 256 \cdot 32$	$2 \cdot 1 \cdot 6^2 \cdot 256 \cdot 32$	589,824
Conv 9	$1 \cdot 1 \cdot 6^2 \cdot 256 \cdot 16$	$1 \cdot 6^2 \cdot 256 \cdot 16$	$2 \cdot 1 \cdot 6^2 \cdot 256 \cdot 16$	294,912
Conv 10	$5^2 \cdot 6^2 \cdot 16 \cdot 32$	$24 \cdot 6^2 \cdot 16 \cdot 32$	$(2 \cdot 5^2 - 1) \cdot 6^2 \cdot 16 \cdot 32$	903,168
Conv 11	$3^2 \cdot 6^2 \cdot 96 \cdot 128$	$8 \cdot 6^2 \cdot 96 \cdot 128$	$(2 \cdot 3^2 - 1) \cdot 6^2 \cdot 96 \cdot 128$	7,520,256
Total	**658,126,368**	**624,473,856**	**1,282,600,224**	**1,282,600,224**

Table 3.12: computations involved in convolutional layers of sub network

Layer	**Multiplications**	**Additions**	**Total Computations**
Conv1	$11 \times 11 \times 55 \times 55 \times 3 \times 96$	$(11 \times 11 - 1) \times 55 \times 55 \times 3 \times 96$	$(2 \times 11 \times 11 - 1) \times 55 \times 55 \times 3 \times 96$
Conv2	$5 \times 5 \times 27 \times 27 \times 48 \times 256$	$(5 \times 5 - 1) \times 27 \times 27 \times 48 \times 256$	$(2 \times 5 \times 5 - 1) \times 27 \times 27 \times 48 \times 256$
Conv3	$3 \times 3 \times 13 \times 13 \times 256 \times 384$	$(3 \times 3 - 1) \times 13 \times 13 \times 256 \times 384$	$(2 \times 3 \times 3 - 1) \times 13 \times 13 \times 256 \times 384$
Conv4	$3 \times 3 \times 13 \times 13 \times 192 \times 384$	$(3 \times 3 - 1) \times 13 \times 13 \times 192 \times 384$	$(2 \times 3 \times 3 - 1) \times 13 \times 13 \times 192 \times 384$
Conv5	$3 \times 3 \times 13 \times 13 \times 192 \times 256$	$(3 \times 3 - 1) \times 13 \times 13 \times 192 \times 256$	$(2 \times 3 \times 3 - 1) \times 13 \times 13 \times 192 \times 256$
Total	**651,767,328**	**618,575,616**	**1,270,342,944**

3.10.2 Determination of computations in AlexNet ReLU layers

The rectification and linearization unit there are no mathematical multiplications or additions involved. The ReLU is an activation function that applies simple thresholding operation, replacing negative values with zeros and positive values unchanged. In the ReLU layer, its activation function will be applied to each neuron input and compares the value. Once the input neuron is compared and if the input value is positive the input value is unchanged. If the input value is negative, then it is multiplied by zero resulting in a zero value which is discarded. Since the net effect of the multiplication by zero has no change in the output it is considered as if no multiplication is involved. Therefore, the computations involved in ReLU layer of a CNN expressed in Equation 3.60 for an input feature map of size W_2 and H_2 with depth K, By applying the Equation 3.60 to the transfer learned AlexNet the computations from the ReLU layers were obtained and presented in Table 3.13.

Table 3.13: computations from ReLU layers of AlexNet

Layer	**Computations (Expression)**	**Total**
ReLU1	$55 \times 55 \times 96$	290,400
ReLU2	$27 \times 27 \times 256$	186,624
ReLU3	$13 \times 13 \times 384$	64,896
ReLU4	$13 \times 13 \times 384$	64,896
ReLU5	$13 \times 13 \times 256$	43,264
Total		**650,080**

Similarly, by applying Equation 3.60 to the cascaded network, computations from the ReLU layers were determined. They were separately presented for main network and sub network in Table 3.14 and Table 3.15 respectively.

Table 3.14: computations from ReLU layers of Main network

Layer	Computations (Expression)	Total
ReLU1	$55 \times 55 \times 96$	290,400
ReLU2	$27 \times 27 \times 256$	186,624
ReLU3	$13 \times 13 \times 384$	64,896
ReLU4	$13 \times 13 \times 384$	64,896
ReLU5	$13 \times 13 \times 256$	43,264
ReLU6	$6 \times 6 \times 96$	3,456
ReLU7	$6 \times 6 \times 32$	1,152
ReLU8	$6 \times 6 \times 64$	2,304
ReLU9	$6 \times 6 \times 16$	576
ReLU10	$6 \times 6 \times 128$	4,608
ReLU11	$6 \times 6 \times 32$	1,152
Total		**663,328**

Table 3.15: computations from ReLU layers of sub network

Layer	Computations (Expression)	Total
ReLU1	$55 \times 55 \times 96$	290,400
ReLU2	$27 \times 27 \times 256$	186,624
ReLU3	$13 \times 13 \times 384$	64,896
ReLU4	$13 \times 13 \times 384$	64,896
ReLU5	$13 \times 13 \times 256$	43,264
Total		**650,080**

3.10.3 Determination of computations in AlexNet Maxpool layers

The purpose of using a maxpool layer in a CNN architecture is to down sample the input feature map prior to applying it to the next immediate layer. In involves sliding a fixed size kernel over the input feature map and selecting the maximum value within each window. The kernel is moved over the input feature map with a specific step known as stride. Thus, the number of output regions are calculated using equation 8. With every stride of the kernel, the maximum value of the corresponding region

becomes the output value of the feature map. By choosing the maximum value within the feature map, it reduces the spatial dimension and thereby sparce connections of the neurons. However, since the maxpool operation involves selection of maximum value within the region, it doesn't involve multiplications. Also, there are no additions directly involved within the pooling operation since there are no modifications to the output feature map values propagated from the input feature map. During the determination of the maximum value, indirect memory related additions are taken place to store the maximum value. Assuming an input feature map of size $W_2 \times H_2$ with a depth of K is to be applied with kernel of size $U \times V$ is moved across the input feature map with stride S. Number of pooling regions can be determined by applying Equation 3.63. By applying the Equation 3.63 to the transfer learned AlexNet, total number of computations from each Maxpool layer was calculated and tabulated in Table 3.16.

Table 3.16: computations from Maxpool layers of Alexnet

Layer	**Computations (Expression)**	**Total**
MaxPool 1	$27 \times 27 \times 3 \times 3 \times 96$	629,856
MaxPool 2	$13 \times 13 \times 3 \times 3 \times 256$	389,376
MaxPool 3	$6 \times 6 \times 3 \times 3 \times 256$	82,944
Total		**1,102,176**

Similarly, by applying Equation 3.63 to the cascaded network, computations from the Maxpool layers were determined. They were separately presented for main network and sub network in Table 3.17 and Table 3.18 respectively.

Table 3.17: computations from Maxpool layers of Main network

Layer	**Computations (Expression)**	**Total**
MaxPool 1	$27 \times 27 \times 3 \times 3 \times 96$	629,856
MaxPool 2	$13 \times 13 \times 3 \times 3 \times 256$	389,376
MaxPool 3	$6 \times 6 \times 3 \times 3 \times 256$	82,944
MaxPool 4	$6 \times 6 \times 3 \times 3 \times 256$	82,944
Total		**1,185,120**

Table 3.18: computations from Maxpool layers of sub network

Layer	**Computations (Expression)**	**Total**
MaxPool 1	$27 \times 27 \times 3 \times 3 \times 96$	629,856
MaxPool 2	$13 \times 13 \times 3 \times 3 \times 256$	389,376
MaxPool 3	$6 \times 6 \times 3 \times 3 \times 256$	82,944
Total		**1,102,176**

3.10.4 Determination of computations in AlexNet Fully connected layers

A fully connected layer is an essential part of convolutional neural network also known as dense connections where neurons form sparse connections are connected to the neurons in the dense layer. At the fully connected layers, outputs from the preceding convolutional layers are taken for regression tasks to perform classification by learning patterns. There are two important tasks performed at the fully connect layer termed flattening and weighted sum activation. In flattening stage, the outputs from the preceding layer essentially 3-dimensional feature map, are flattened into a 1-dimensional vector. Each neuron is then appended with weight and bias is added. For input vector of size (J) with bias (b), the computations can be derived as described in previous Equations. Since each neuron in the fully connected layer is connected to every element in the input, dot product requires (J) multiplications and $(J-1)$ additions. Therefore, the number of computations is approximately $2J$. If bias of b present, the total

number of computations per neuron is given in Equation 3.65. By applying Equation 3.65 to the transfer learned AlexNet, total number of multiplications additions and hence computations were determined and tabulated in Table 3.19. By applying Equation 3.65 to the subnetwork, the number of computations were calculated and presented in Table 3.20. Similarly for the main network computations were calculated and tabulated in Table 3.21 respectively.

Table 3.19: computations from Fully connected layers of Alex net

Layer	**Multiplications**	**Additions**	**Total Computations**
Fully Connected 1	9216×4096	$(9216 - 1 + 1) \times 4096$	75,497,472
Fully Connected 2	4096×4096	$(4096 - 1 + 1) \times 4096$	33,554,432
Fully Connected 3	4096×36	$(4096 - 1 + 1) \times 36$	294,912
Total	**54,673,408**	**54,673,408**	**109,346,816**

Table 3.20: Fully Connected Layer Computations

Layer	**Multiplications**	**Additions**	**Total Computations**
Fully Connected	256×4	$(256 - 1 + 1) \times 4$	2,048
Total	1,024	1,024	

Table 3.21: computations from Fully connected layers of main network

Layer	**Multiplications**	**Additions**	**Total Computations**
Fully Connected	256×6	$(256 - 1 + 1) \times 6$	3,072
Total	1,536	1,536	

By considering all the computations stated above, a summary of the number of computations is presented in Table 3.22.

Table 3.22: comparison of computations

Layer	**Cascaded Network**	**AlexNet**
Convolutional	2,552,943,168	1,270,342,944
ReLU	1,313,408	650,080
Maxpool	2,287,296	1,102,176
Fully Connect	5,120	109,346,816
Total	2,556,548,992	1,381,442,016

3.10.5 Execution time measurement

Since broader elaboration presented on previous chapters in theoretically but also need to present them in practical. As stated above, deep neural networks involve a larger number of mathematical computations in various layers of their architecture. The time taken by the neural network model to make a classification can be estimated with the computer involved for the classification process. It is noted that the execution time is a rough estimate but not a precise time measurement since the time is dependent on factors such as CPU clock frequency, CPU cores, Memory Bandwidth [128]. In convolutional layers of a network model, large number of multiplications and additions are repeated, which contribute to high latency as well as more power consumption [112]. Resource constraints in mobile computational platforms thus a significant challenge during the implementation of CNN models as computational complexity and energy efficiency should be critically considered. In addition to these challenges, since the CNN models involve a larger number of learnable parameters associated with as mentioned in section 3.6, it is required large memory to store the instantaneous data [112] [129]. Overseeing these constraints execution time estimation was conducted for the proposed cascaded network model

and standard AlexNet. The AlexNet model was re-trained using transfer learning with the same dataset used for the cascaded network. For different object classes both the networks were trained using high performance computers and stored with respective class names for later experimental purposes. Time measurement was conducted on different computers to observe the characteristics of each CNN model. A set of images were used for both network models to test the execution time taken by each network respectively. Each test image was taken as an input image to the network and executed the model to pass through all the layers until finally construct the class output. This process was repeated five times with an interval of 2 seconds and those five results were recorded and average was taken as the execution time for the corresponding image input. Similarly, the execution time measurement was conducted for all the test images created at the beginning. Part of the execution time results for cascaded network and transfer learned AlexNet are presented in Table 3.23 and Table 3.24 respectively.

Table 3.23: cascaded network execution time for different classes

Test Object	**15 classes**	**25 classes**	**40 classes**	**50 classes**	**100 classes**	**200 classes**
cleaning machine2.jpg	0.056511	0.056850	0.058375	0.057876	0.058957	0.060531
scanning machine1.jpg	0.059168	0.059370	0.059931	0.060483	0.062034	0.063023
scanning machine3.jpg	0.056495	0.056957	0.057696	0.058866	0.059594	0.060255
buggy cart 4.png	0.058392	0.058510	0.058583	0.059250	0.061399	0.061164
cat1.jpg	0.055661	0.055704	0.056601	0.058093	0.058458	0.059062
bird1.jpg	0.054368	0.055072	0.055469	0.055898	0.056488	0.058256
butterfly 3.jpg	0.058137	0.058461	0.058952	0.059811	0.060274	0.061036
dog31.jpg	0.061599	0.061649	0.061896	0.062368	0.063129	0.062002
pig5.jpg	0.059788	0.060251	0.060735	0.062368	0.062767	0.063893
pig1.jpg	0.059762	0.060034	0.060610	0.061130	0.061818	0.062713
pig14.jpg	0.055840	0.056087	0.056715	0.056966	0.057578	0.061265
buggy44.jpg	0.056159	0.056337	0.056727	0.056996	0.057801	0.059063
buggy36.jpg	0.064847	0.065471	0.065912	0.066590	0.067532	0.068970
stroller2.jpg	0.057536	0.057799	0.058371	0.058878	0.059553	0.062583
stroller3.jpg	0.058304	0.058654	0.059345	0.060833	0.061402	0.062253
trolley14.jpg	0.058017	0.058377	0.058664	0.059369	0.060260	0.061096
trolley20.jpg	0.055750	0.061071	0.061620	0.062176	0.062567	0.064581
Average time	0.058020	0.058627	0.059188	0.059879	0.060683	0.061867

Table 3.24: transfer learned AlexNet execution time for different classes.

Test Object	**15 classes**	**25 classes**	**40 classes**	**50 classes**	**100 classes**	**200 classes**
Cleaning machine2.jpg	0.046271	0.046727	0.046825	0.048862	0.059822	0.067615
scanning machine1.jpg	0.044271	0.045267	0.047858	0.047858	0.058538	0.068017
scanning machine3.jpg	0.044641	0.045971	0.047769	0.048769	0.055598	0.066799
buggy cart 4.png	0.044491	0.046691	0.045232	0.046232	0.058611	0.067828
cat1.jpg	0.047282	0.045688	0.046283	0.049283	0.057879	0.067842
bird1.jpg	0.043623	0.045672	0.047216	0.049216	0.056779	0.065379
butterfly3.jpg	0.043896	0.046077	0.047461	0.047461	0.058692	0.068572
dog31.jpg	0.046203	0.047541	0.048672	0.049662	0.056286	0.067015
pig5.jpg	0.046957	0.045911	0.047535	0.047975	0.058925	0.069996
pig14.jpg	0.045608	0.046628	0.047942	0.049342	0.055205	0.065876
buggy44.jpg	0.045366	0.046832	0.047954	0.048688	0.057669	0.066504
buggy36.jpg	0.045687	0.046793	0.045854	0.047863	0.055624	0.066221
stroller2.jpg	0.045441	0.045911	0.047165	0.048261	0.055941	0.065783
strroller3.jpg	0.046524	0.046631	0.047547	0.049547	0.058215	0.067976
trolley14.jpg	0.046633	0.047438	0.047137	0.047911	0.059405	0.068392
trolley20.jpg	0.045842	0.046164	0.046915	0.048527	0.057822	0.067615
Average time	0.045546	0.046371	0.047210	0.048466	0.057563	0.067339

Considering the average execution times from both network models, a comparison was presented in Figure 3.4 with respective the number of classes.

Referring to the execution time comparison presented in Figure 3.26, it was observed that as the number of classes were increased the execution time for the AlexNet increased gradually with lesser rate of increment at initially for lower output classes which then increased over the number of classes. Cascaded network on the other hand recorded comparatively higher execution time of 58ms at the beginning for low output classes which is 13ms higher than AlexNet. It was observed that cascaded network execution time increases over the number of output classes. However, the rate at which it increases is less compared to AlexNet. It was also observed that the execution time of both networks becomes equal to approximately 135 classes. The computer on which we conducted the test has a base frequency of $2.4GHz$. If runs at the base frequency the single machine cycle time is 4.1666×10^{-10} seconds. Since a single instruction execution requires 4 machine cycles, the time to execute one computation is 1.041666×10^{-10} seconds. Since there are 4 cores in the processor, the execution time is $2.60416668 \times 10^{-11}$ seconds. To execute cascaded network with 25 classes, there are 2556548992 computations to be performed. Hence the maximum possible time required to execute is 0.066576797 seconds. This is within the practical execution time observed during the execution of the network model.

3.11 Chapter Summary

This chapter outlines the comprehensive methodological framework and system design used to meet the research objectives of developing a lightweight

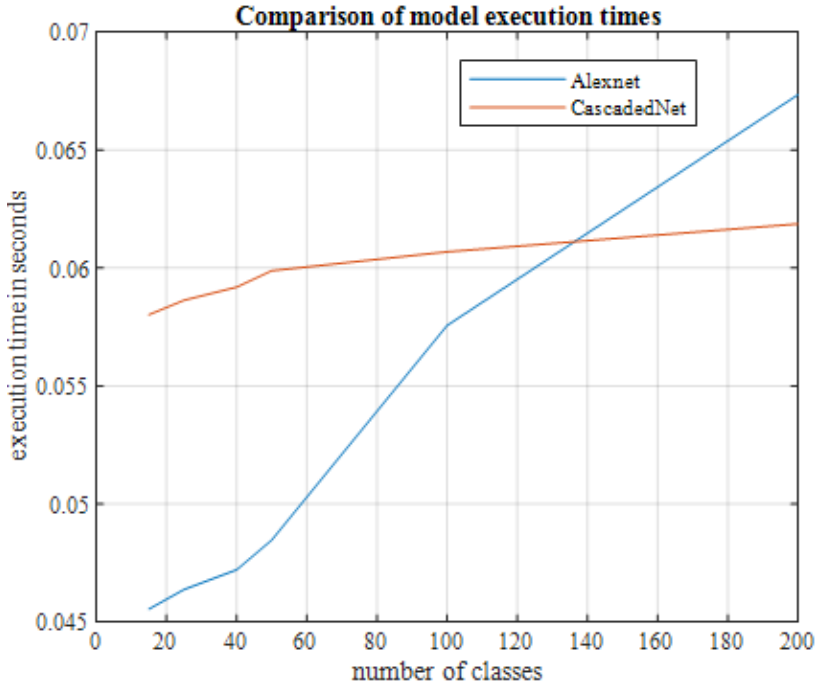


Figure 3.26: comparison of CNN execution times

cascaded convolutional neural network (CNN) for real-time image classification on edge devices. The research methodology was structured to combine theoretical design with experimental validation, ensuring that the proposed model achieves both computational efficiency and high classification accuracy.

The chapter started by outlining the overall research workflow, which guided the logical progression from defining the problem to implementing and evaluating the model. A system flow chart illustrated the two-stage cascaded CNN architecture, consisting of a main network for primary classification and a secondary network for confidence-based refinement. This design allows for the selective activation of deeper layers only when necessary, thereby minimizing inference time while maintaining robust accuracy. The chapter also provided a detailed explanation of various methods and optimization strategies, including efficient layer design, inverted residual blocks, quantization-aware computation, layer fusion, and pruning. These architectural enhancements collectively contribute to reducing model complex-

ity, improving memory utilization, and achieving faster inference, making it suitable for embedded and mobile platforms, such as the Raspberry Pi. Furthermore, the chapter described the experimental methodology, which included model training, dataset preparation, and hardware deployment. To ensure an objective assessment of the proposed model's performance against existing CNN architectures, benchmarking and performance evaluation metrics—such as accuracy, inference time, and energy consumption—were clearly defined.

The methodology chapter provides a thorough foundation for the experimental investigations described in the following chapter. The systematic approach used in this section ensures that the proposed cascaded CNN design is both theoretically sound and practically validated. This forms the basis for the subsequent results and analysis, which demonstrate the design's efficiency, scalability, and applicability to real-time edge AI systems.

CHAPTER 4

RESULTS AND DISCUSSION

Contents

4.1 Introduction to the Chapter **154**

4.2 Results of the cascaded network model **155**

4.2.1 Computations in main network 155

4.2.2 Impact on number of classes 157

4.3 Performance analysis of sub network **161**

4.4 Combined network's performance **165**

4.5 Classification accuracies **170**

4.6 CNN Model Benchmarking **181**

4.7 Chapter Summary **186**

4.1 Introduction to the Chapter

This chapter presents the results of the implementation, training, and evaluation of the proposed cascaded convolutional neural network (CNN) model developed using MATLAB. The primary goal of this analysis is to assess the performance of the proposed architecture in terms of classification accuracy, inference time, and computational efficiency when deployed on embedded and edge computing platforms. The results of both the main CNN and the secondary refinement network were analyzed to validate the effectiveness of the cascaded design. Additionally, a comparative assessment is conducted against baseline models, including AlexNet and other conventional CNN architectures, to demonstrate the improvements achieved through the optimized multi-stage approach. Furthermore, the chapter discusses how various optimization strategies—such as efficient layer design, inverted residual blocks, pruning, and hardware-aware computation contribute to the overall performance gains. The results are interpreted in relation to the research objectives and hypotheses, emphasizing the model's suitability for real-time robotic guidance and edge-based image classification tasks.

The remainder of this chapter is organized as follows: section 4.2 presents the results of the cascaded model, focusing on the number of computations and the impact on the number of classes for the overall performance of the proposed cascaded CNN, highlighting accuracy trends and convergence behavior. Section 4.3 analyzes the performance of the subnetwork model. Section 4.4 discusses the combined network performance and compares the proposed model's results with those of baseline networks such as AlexNet and MobileNet. Section 4.5 addresses the classification accuracies of both the main network and the sub network, as well as the impact of architectural optimizations including efficient layer design, pruning, and

hardware-aware design on the final outcomes. Finally, section 4.6 summarizes the key findings and evaluates them in relation to the research objectives and hypotheses established earlier in the thesis.

4.2 Results of the cascaded network model

The results of the proposed cascaded network model are described in two sections, section 4.4 for the main network's performance and section 4.5 for the sub network. In this section the results are presented as performance results and analysis and the computation related to the main network. The computations involved in every layer of the main network were manually calculated.

4.2.1 Computations in main network

In the main convolutional neural network, there are eleven convolutional layers, four max pooling layers and one fully connected layer present. Since the above-mentioned layers involve complex computations, we focus only on those layers that directly contribute to the execution time. The max pooling operation involves sliding windows operations, but does not contribute to the learning process. Hence, the max pooling layer is considered as a non-learnable layer. The parameter calculation involves the convolutional and fully connected layers. The detailed methodology is explained in section 3.6. The total learnable parameters are presented in Table 4.1.

The formula for the mathematical multiplications and additions is explained in section 3.6. By using the formula, the total number of mathematical additions are 105489984 and 118016928 multiplications are to be

Table 4.1: Learnable Parameters of Main Network

Layer	**Learnable Parameters**
Conv1	34,944
Conv2	307,456
Conv3	885,120
Conv4	663,936
Conv5	442,624
Conv6	8,824
Conv7	16,448
Conv8	24,672
Conv9	110,720
Conv10	4,112
Conv11	12,832

performed in single execution cycle of the main network model. The theoretical data associated with the convolutional layer are presented in Table 4.2.

Table 4.2: Summary of Measurable Quantities

Measurable	**Quantity**
Total learnable parameters	2,511,688
Total number of multiplications	118,016,928
Total number of additions	105,489,984
Execution time for multiplications (s)	0.07376058
Execution time for additions (s)	0.065936865
Practical execution time (s)	0.207778
Practical time difference (s)	0.06

Referring to the time values in Table 4.2, it is observed that the difference in the practical timing value of 0.06 seconds. This time difference is due to the operations involved in ReLU, Maxpool and fully connected layers. The computations involved in these layers are presented in section 3.6. However, the execution time consumptions of the ReLu, Maxpool and Fully connected layers are presented in Table 4.3.

Referring to the quantities in Table 4.3, the time difference of 0.06 seconds

Table 4.3: Execution Time for Different Layers

Execution Layer	**Time (seconds)**
Rectified Linear Unit (ReLU)	0.0029864
Maxpool Unit	0.005447645
Fully Connected	0.0530841
Total Time	**0.061**

is introduced by the ReLU, Maxpool and Fully connected layers which is the time difference in the practical values of time difference of the complete execution of the network model. However, referring to the values in both Table 4.1. and 4.2, it is observed that there exists a time difference of 0.06 seconds as presented in Table 4.3.

As explained in section 4.4 the time difference between the practical execution and theoretical execution time was 0.016 seconds. This was observed in the very first layer of the network, the image input layer. Therefore, the time differences are balanced, and thus theoretical values are in agreement with the practical values as mentioned in Table 4.4.

Table 4.4: Comparison of theoretical and practical execution time for main network

Measurable	**Main Network (sec)**
Practical execution time	0.207778
Theoretical (convolutions)	0.139697445
Theoretical (ReLU + Maxpool + FC)	0.0549706
Image input layer execution time	0.016

4.2.2 Impact on number of classes

It is worth analyzing that the networks' performances change as the number of classes are changed. This research was conducted using different number of classes beginning with 4 classes. The main network was trained for 4 classes initially and the number was increased substantially in stages, the

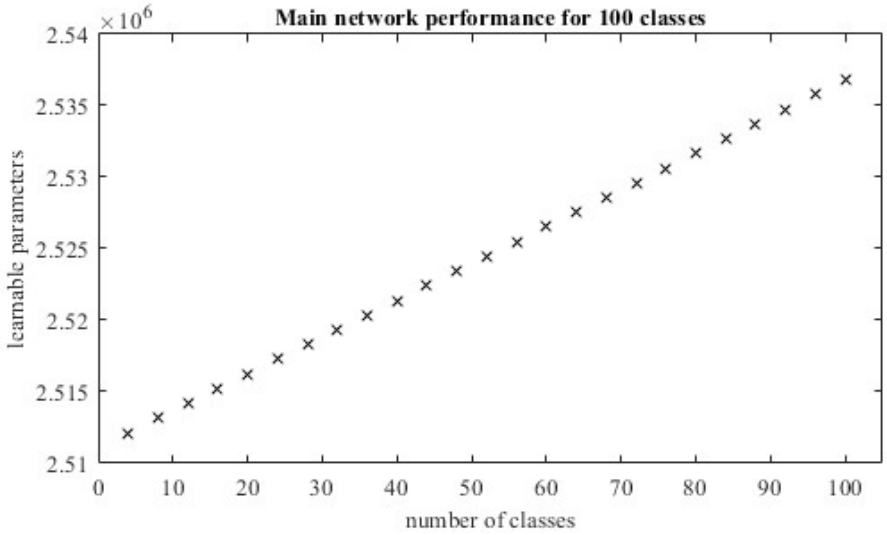


Figure 4.1: Main network performance for number of classes up to 100

performances were recorded and plotted in Figure 3.23. Similarly, the sub network was initially trained for 5 classes and consequently increased up to 100 with groups of five in the first stage which is presented in Figure 4.3. During the second stage, it was required to compare the results for 1000 classes, by which the standard AlexNet network had been trained for 1000 object classes using CIFAR dataset from MNIST challenge. Therefore, both the network models and the subnetwork were also trained for 1000 object classes in groups of 100. These results are presented in Figure 4.4 and for the main network model's results are presented in Figure 4.5.

Referring to Figure 4.1 which represents the relationship between learnable parameters vs number of classes in the main network. A smaller number of classes ranging from four up to one hundred with a group of four were tested, and the number of learnable parameters calculated for each class were represented by each dot in the figure. A linear relationship can be observed over the increasing number of classes.

Figure 4.2 depicts the main network architecture's performance for a larger number of classes ranging from 100 to 1000. The tests were carried out in

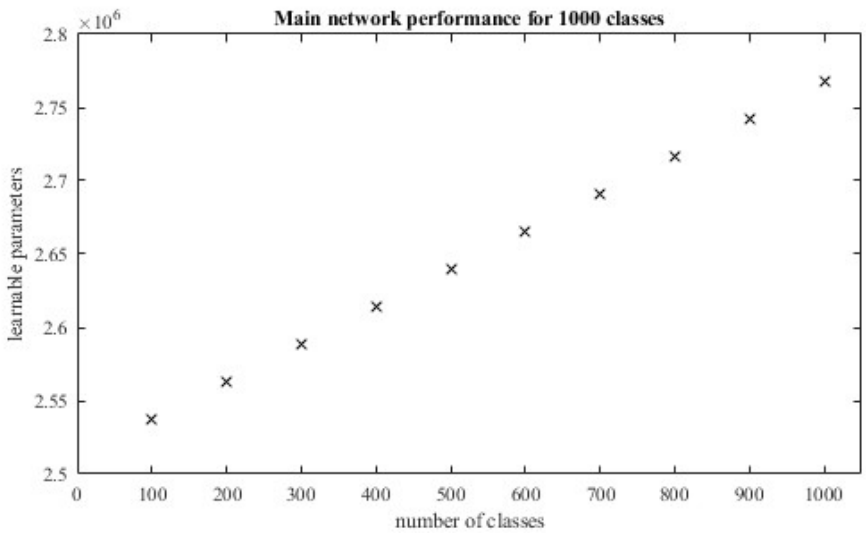


Figure 4.2: Main network performance for number of classes from 100 to 1000

steps of 100 where the maximum number of classes was limited to 1000 to compare for analysis with standard AlexNet network which was also trained for same number of classes. Learnable parameters increased with the increasing number of classes exhibiting a linear relationship.

Table 4.5 presents the total number of learnable parameters in AlexNet and the proposed cascaded network as the number of output classes increases from 25 to 1000. The results demonstrate a significant difference in model scalability between the two architectures. The state-of-the-art AlexNet shows a consistent increase in the parameter count with increasing class complexity. Starting with 56,970,649 parameters for 25 classes, AlexNet reaches 60,965,224 for 1000 classes, resulting in an increase of approximately 4 million parameters (7.0%). This growth is primarily due to the architecture's fully connected classification layers, which scale linearly with the number of target classes.

In contrast, the cascaded network maintains a nearly constant parameter count across all configurations, ranging from 4,892,538 to 4,916,953 parameters, an increase of only 24,415 parameters (0.5%). This minimal

dependence on the class count indicates that the majority of the network's parameters reside in shared, class-independent feature extraction modules, while the classification head is designed to scale efficiently.

In particular, the cascaded network operates with approximately 92% fewer parameters than the state-of-the-art AlexNet, even at the highest class count of 1000 object classes, without compromising architectural expressiveness. These findings highlight the proposed model's superior parameter efficiency, making it suitable for large scale classification tasks where model compactness and scalability are critical. In addition, it demonstrates suitability for resource constrained environments, such as edge devices, enabling efficient utilization of computational resources while producing accurate classification results.

Table 4.5: Classes combination and parameters for Alexnet and cascaded network

Number of Classes	AlexNet	Cascaded Network
25	56,970,649	4,892,538
50	57,073,074	4,893,823
75	57,175,499	4,895,108
100	57,277,924	4,896,393
150	57,482,774	4,898,963
200	57,687,624	4,901,533
250	57,892,474	4,904,103
300	58,097,324	4,906,673
350	58,302,174	4,909,243
400	58,507,024	4,911,813
450	58,711,874	4,948,903
500	58,916,724	4,950,188
550	59,121,574	4,951,473
600	59,326,424	4,952,758
650	59,531,274	4,954,043
700	59,736,124	4,955,328
750	59,940,974	4,956,613
800	60,145,824	4,957,898
850	60,350,674	4,959,183
900	60,555,524	4,960,468
950	60,760,374	4,961,753
1000	60,965,224	4,916,953

4.3 Performance analysis of sub network

This section presents results for the sub network model. The computations involved in every layer in the sub network are manually calculated and the results from the practical implementation are compared with the theoretical values. In the sub network model, there are five convolutional layers, three

max pooling layers and a fully connected layer are present. Total number of learnable parameters, complex multiplications and additions are calculated and presented in Table 4.6.

Table 4.6: Computations and learnable parameters in sub network

Layer	**Learnable Parameters**	**Num. Additions**	**Num. Multiplications**
Conv1	34,944	95,832,000	105,415,200
Conv2	307,456	7,464,960	9,331,200
Conv3	885,120	225,792	338,688
Conv4	663,936	225,792	338,688
Conv5	442,624	150,528	225,792

Using the mathematical model explained in section 3.6, it was found that a total of 115649568 number of complex multiplications and 103899072 additions are to be performed in single execution cycle. The time taken for these operations are measured in real time and compared with the theoretical values. Table 4.7 presents the computed values.

Table 4.7: Secondary network parameters

Measurable	**Quantity**
Total learnable parameters	2,334,080
Total number of multiplications	115,649,568
Total number of additions	103,899,072
Execution time for multiplications	0.07228 seconds
Execution time for additions	0.06494 seconds
Practical execution time	0.20236 seconds
Practical time difference	0.06 seconds

Referring to the timing values mentioned in Table 4.8, it is noted that there is a time difference between the practical time value and the theoretical value of 0.06 seconds. This is the same time difference as in the case of section 4.2. The time difference is due to the processing of the

intermediate layers such as ReLU, Maxpool and fully connected. However, when we consider the time difference between practical implementation vs theoretical it is noted that there is a small difference in theorical time. Thus, it was required to further analyze the time profile. It was clearly seen that a considerable amount of time is taken by the image input layer as it must perform preprocessing of the input data so that the information can be passed through the sub sequent layers of the network. The image input layer, which involves two mathematical operations such as normalization and resizing, thus significantly contributes to the execution time of the network model. Since the image input sizes remain the same and equal dimensional for both network models, the time value remains same for both the networks.

Table 4.8: Comparison of main and secondary network

Measurement	**Main network**	**Sub network**
Practical execution time	0.207778	0.202361
Theoretical (convolutions)	0.1397	0.1372
Theoretical (ReLU + Maxpool + Fully connect)	0.05497	0.05406
Image input layer execution time	0.016	0.016

Performance of the sub network model was tested in two stages by dividing the complete dataset into groups of 5 and 100 respectively. The network model trained all datasets ranging from 5 to 100 classes in the first stage and 100 to 1000 classes in the second stage. The results were tabulated, and the graphs were plotted in Figure 4.3 and 4.9.

The sub network performance should also be considered to make a better comment on the novel network architecture's performance. Figure 4.3 presents the variation of learnable parameters as the number of classes were increased. It was observed that a linear relationship between number of classes and the learnable parameters. The number of classes in the sub network was changed from five to hundred in steps of five and the learnable

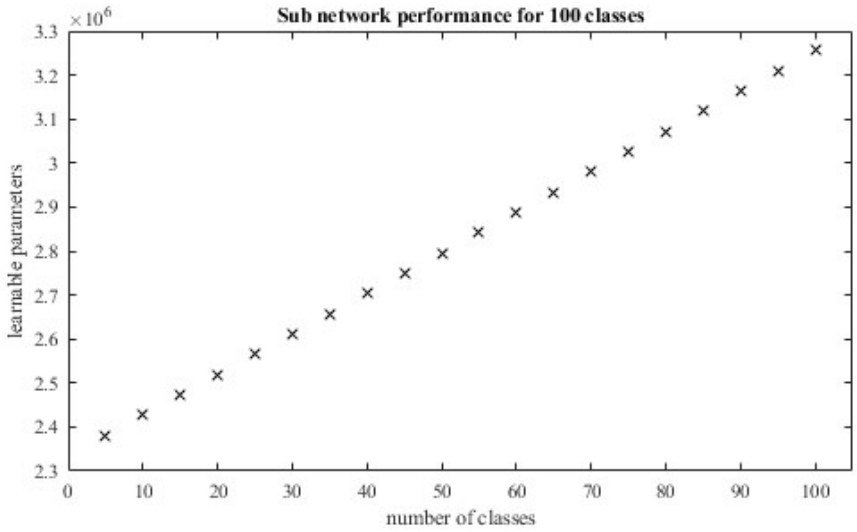


Figure 4.3: Sub network performance for the classes from 5 to 100

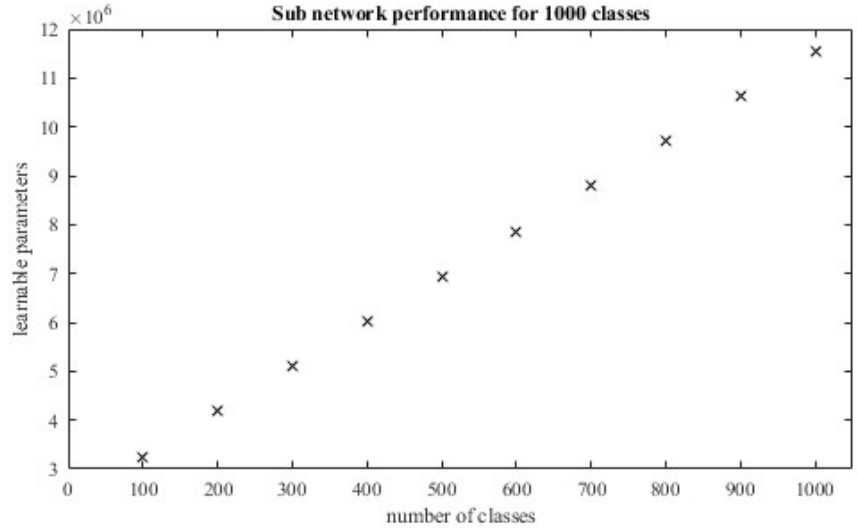


Figure 4.4: Sub network performance for the classes from 100 to 1000

parameters calculated were depicted in Figure 4.3.

Figure 4.4 represents the relationship between learnable parameters and number of classes. The number of classes were increased from one hundred to thousand in steps of hundred to compare the performance against the standard AlexNet. A linear relationship was observed as the number of classes were increased gradually in steps as like the previous observations. From the results plotted it can also be stated that there exists a linear relationship between the number of classes and learnable parame-

ters. Mathematically, as the number of classes were increased the complex computations involved are also increased and thus linearity observed.

4.4 Combined network's performance

The individual performance of the main and sub network models was explained and presented in sections 4.1 and 4.2, respectively. Since this research was intended to produce a single network model, both network models were combined with the combination logic unit through which the flow of direction is controlled. The overall performance of the network model is presented in this chapter. As discussed in the previous chapter, the network model has two sections main network and sub network. Durning the full network execution the test was conducted in such a way that execution time for each network was recorded over the number of classes considering both networks. Ideally, the number of classes for the main network was recorded simultaneously for the sub network resulting in a two-dimensional matrix of data. From the data recorded, the number of classes was offering different options for the same number. As an example, for 100 object classes, a combination of main and sub network proposes as 10 main classes and 10 sub classes, 20 main classes and 5 sub classes, 5 main classes and 20 sub classes etc. equivalent execution time matrix also created matching with the number of classes in both network models. For a selected class of interest, all possible combinations were recorded, and their corresponding execution time values were also recorded. Out of the possible combinations, the minimum execution time was determined, and the combination was selected. The minimum execution time recorded was plotted against the number of classes to present the performance of the new network model. The results were compared with standard AlexNet.

The selection of the number of classes from the main and sub networks is presented in Table 4.9.

Table 4.9: Main network and sub network classes combination

		main network									
		5	10	15	20	25	30	35	40	45	50
sub network	5	25	50	75	100	125	150	175	200	225	250
	10	50	100	150	200	250	300	350	400	450	500
	15	75	150	225	300	375	450	525	600	675	750
	20	100	200	300	400	500	600	700	800	900	1000
	25	125	250	375	500	625	750	875	1000	1125	1250
	30	150	300	450	600	750	900	1050	1200	1350	1500
	35	175	350	525	700	875	1050	1225	1400	1575	1750
	40	200	400	600	800	1000	1200	1400	1600	1800	2000
	45	225	450	675	900	1125	1350	1575	1800	2025	2250
	50	250	500	750	1000	1250	1500	1750	2000	2250	2500
	55	275	550	825	1100	1375	1650	1925	2200	2475	2750
	60	300	600	900	1200	1500	1800	2100	2400	2700	3000
	65	325	650	975	1300	1625	1950	2275	2600	2925	3250
	70	350	700	1050	1400	1750	2100	2450	2800	3150	3500
	75	375	750	1125	1500	1875	2250	2625	3000	3375	3750
	80	400	800	1200	1600	2000	2400	2800	3200	3600	4000
	85	425	850	1275	1700	2125	2550	2975	3400	3825	4250
	90	450	900	1350	1800	2250	2700	3150	3600	4050	4500
	95	475	950	1425	1900	2375	2850	3325	3800	4275	4750
	100	500	1000	1500	2000	2500	3000	3500	4000	4500	5000

(a) Part A

		main network									
		55	60	65	70	75	80	85	90	95	100
sub network	5	275	300	325	350	375	400	425	450	475	500
	10	550	600	650	700	750	800	850	900	950	1000
	15	825	900	975	1050	1125	1200	1275	1350	1425	1500
	20	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000
	25	1375	1500	1625	1750	1875	2000	2125	2250	2375	2500
	30	1650	1800	1950	2100	2250	2400	2550	2700	2850	3000
	35	1925	2100	2275	2450	2625	2800	2975	3150	3325	3500
	40	2200	2400	2600	2800	3000	3200	3400	3600	3800	4000
	45	2475	2700	2925	3150	3375	3600	3825	4050	4275	4500
	50	2750	3000	3250	3500	3750	4000	4250	4500	4750	5000
	55	3025	3300	3575	3850	4125	4400	4675	4950	5225	5500
	60	3300	3600	3900	4200	4500	4800	5100	5400	5700	6000
	65	3575	3900	4225	4550	4875	5200	5525	5850	6175	6500
	70	3850	4200	4550	4900	5250	5600	5950	6300	6650	7000
	75	4125	4500	4875	5250	5625	6000	6375	6750	7125	7500
	80	4400	4800	5200	5600	6000	6400	6800	7200	7600	8000
	85	4675	5100	5525	5950	6375	6800	7225	7650	8075	8500
	90	4950	5400	5850	6300	6750	7200	7650	8100	8550	9000
	95	5225	5700	6175	6650	7125	7600	8075	8550	9025	9500
	100	5500	6000	6500	7000	7500	8000	8500	9000	9500	10000

(b) Part B

The corresponding execution time values are presented in Table 4.10 where the minimum value for a specific combination is highlighted.

The overall performance of the proposed network model is presented in the graph given in Figure 4.5.

Table 4.10: Learnable parameter selection for main and sub network classes

			main network learnable parameters									
			5	10	15	20	25	30	35	40	45	50
			2512373	2513658	2514943	2516228	2517513	2518798	2520083	2521368	2522653	2523938
sub network learnable parameters	5	2380165	4892538	**4893823**	4895108	4896393	4897678	4898963	4900248	4901533	4902818	4904103
	10	2426250	**4938623**	4939908	4941193	4942478	4943763	4945048	4946333	4947618	4948903	4950188
	15	2472335	4984708	4985993	4987278	4988563	4989848	4991133	4992418	4993703	4994988	4996273
	20	2518420	5030793	5032078	5033363	5034648	5035933	5037218	5038503	5039788	5041073	5042358
	25	2564505	5076878	5078163	5079448	5080733	5082018	5083303	5084588	5085873	5087158	5088443
	30	2610590	5122963	5124248	5125533	5126818	5128103	5129388	5130673	5131958	5133243	5134528
	35	2656675	5169048	5170333	5171618	5172903	5174188	5175473	5176758	5178043	5179328	5180613
	40	2702760	5215133	5216418	5217703	5218988	5220273	5221558	5222843	5224128	5225413	5226698
	45	2748845	5261218	5262503	5263788	5265073	5266358	5267643	5268928	5270213	5271498	5272783
	50	2794930	5307303	5308588	5309873	5311158	5312443	5313728	5315013	5316298	5317583	5318868
	55	2841015	5353388	5354673	5355958	5357243	5358528	5359813	5361098	5362383	5363668	5364953
	60	2887100	5399473	5400758	5402043	5403328	5404613	5405898	5407183	5408468	5409753	5411038
	65	2933185	5445558	5446843	5448128	5449413	5450698	5451983	5453268	5454553	5455838	5457123
	70	2979270	5491643	5492928	5494213	5495498	5496783	5498068	5499353	5500638	5501923	5503208
	75	3025355	5537728	5539013	5540298	5541583	5542868	5544153	5545438	5546723	5548008	5549293
	80	3071440	5583813	5585098	5586383	5587668	5588953	5590238	5591523	5592808	5594093	5595378
	85	3117525	5629898	5631183	5632468	5633753	5635038	5636323	5637608	5638893	5640178	5641463
	90	3163610	5675983	5677268	5678553	5679838	5681123	5682408	5683693	5684978	5686263	5687548
	95	3209695	5722068	5723353	5724638	5725923	5727208	5728493	5729778	5731063	5732348	5733633
	100	3255780	5768153	5769438	5770723	5772008	5773293	5774578	5775863	5777148	5778433	5779718

(a) Part A

			main network learnable parameters									
			55	60	65	70	75	80	85	90	95	100
			2525223	2526508	2527793	2529078	2530363	2531648	2532933	2534218	2535503	2536788
sub network learnable parameters	5	2380165	4905388	4906673	4907958	4909243	4910528	4911813	4913098	4914383	4915668	4916953
	10	2426250	4951473	4952758	4954043	4955328	4956613	4957898	4959183	4960468	4961753	4963038
	15	2472335	4997558	4998843	5000128	5001413	5002698	5003983	5005268	5006553	5007838	5009123
	20	2518420	5043643	5044928	5046213	5047498	5048783	5050068	5051353	5052638	5053923	5055208
	25	2564505	5089728	5091013	5092298	5093583	5094868	5096153	5097438	5098723	5100008	5101293
	30	2610590	5135813	5137098	5138383	5139668	5140953	5142238	5143523	5144808	5146093	5147378
	35	2656675	5181898	5183183	5184468	5185753	5187038	5188323	5189608	5190893	5192178	5193463
	40	2702760	5227983	5229268	5230553	5231838	5233123	5234408	5235693	5236978	5238263	5239548
	45	2748845	5274068	5275353	5276638	5277923	5279208	5280493	5281778	5283063	5284348	5285633
	50	2794930	5320153	5321438	5322723	5324008	5325293	5326578	5327863	5329148	5330433	5331718
	55	2841015	5366238	5367523	5368808	5370093	5371378	5372663	5373948	5375233	5376518	5377803
	60	2887100	5412323	5413608	5414893	5416178	5417463	5418748	5420033	5421318	5422603	5423888
	65	2933185	5458408	5459693	5460978	5462263	5463548	5464833	5466118	5467403	5468688	5469973
	70	2979270	5504493	5505778	5507063	5508348	5509633	5510918	5512203	5513488	5514773	5516058
	75	3025355	5550578	5551863	5553148	5554433	5555718	5557003	5558288	5559573	5560858	5562143
	80	3071440	5596663	5597948	5599233	5600518	5601803	5603088	5604373	5605658	5606943	5608228
	85	3117525	5642748	5644033	5645318	5646603	5647888	5649173	5650458	5651743	5653028	5654313
	90	3163610	5688833	5690118	5691403	5692688	5693973	5695258	5696543	5697828	5699113	5700398
	95	3209695	5734918	5736203	5737488	5738773	5740058	5741343	5742628	5743913	5745198	5746483
	100	3255780	5781003	5782288	5783573	5784858	5786143	5787428	5788713	5789998	5791283	5792568

(b) Part B

Cascaded network performance for 1000 classes is presented in Figure 4.6. The performance was tested in two stages where lesser number of classes ranging from 5 to 100 in the first stage as presented in Figure 4.5 and 100 to 100 in the second stage as presented in Figure 4.6. From the results a linear relationship observed over the increasing number of classes.

The cascaded network model was compared with one of the conventional neural network models, AlexNet. To balance the test conditions, AlexNet

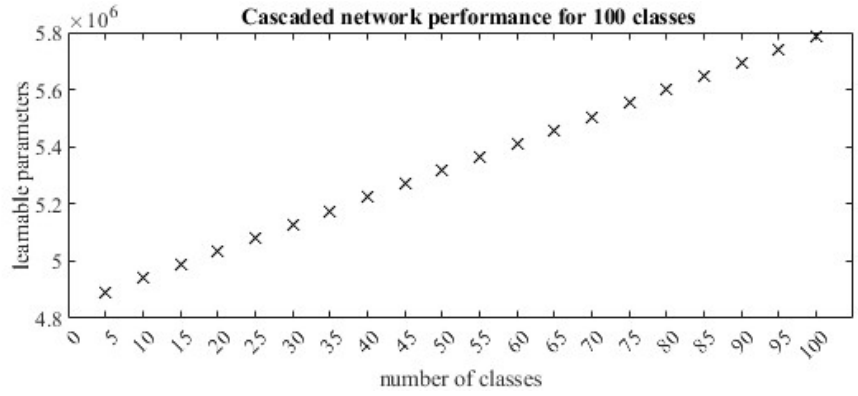


Figure 4.5: Cascaded network performance

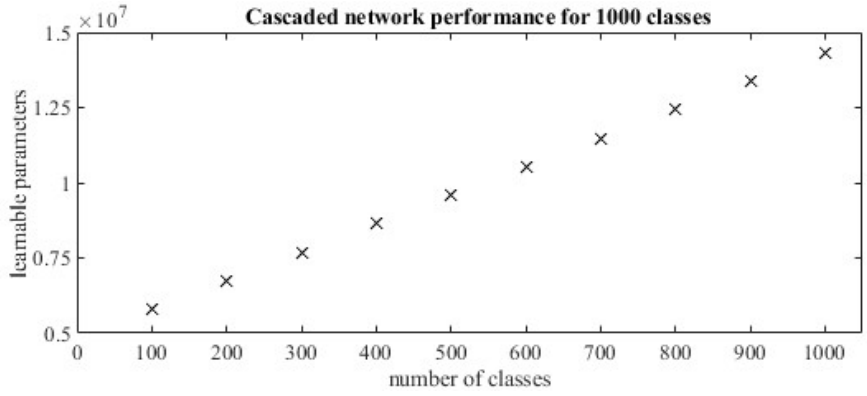


Figure 4.6: Cascaded network performance for 1000 classes

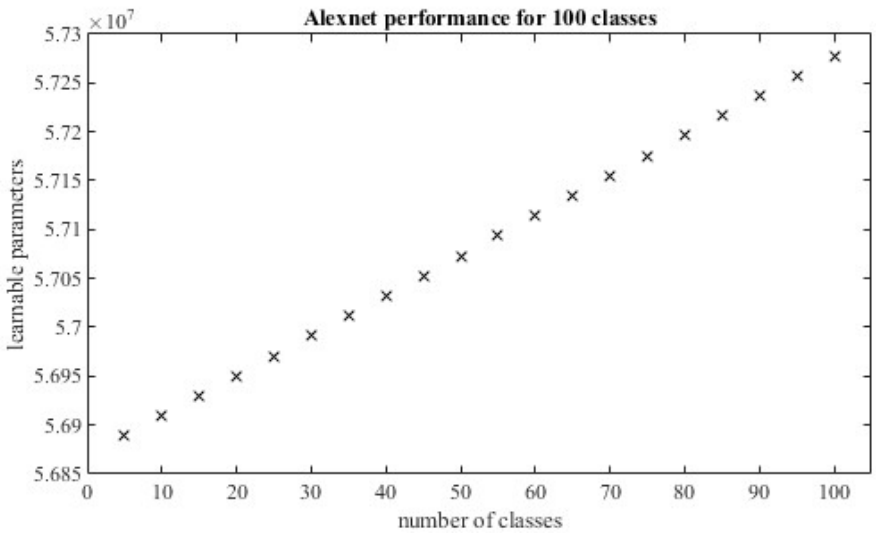


Figure 4.7: AlexNet network performance for number of classes from 5 to 100

was re-trained with the dataset with classes ranging from 5 to 100 in the first stage and 100 to 1000 in second stage. For every number of classes, the learnable parameters were calculated manually. The performance of AlexNet network is presented in Figure 4.7 and 4.8 respectively.

Figure 4.7 depicts the characteristics of standard AlexNet network the learnable parameters variation in accordance with the number of classes. As explained earlier, a linear relationship can be observed between the learnable parameters and number of classes given as in the graph. The test was conducted for different number of classes ranging from five to hundred in steps of five. The corresponding learnable parameters were calculated and plotted against the number of classes to obtain the graph.

The number of classes were increased from hundred to thousand in steps of hundred while observing the learnable parameters. Similar linear relationship was observed for learnable parameters as the number of classes increased. Since the standard AlexNet trained for a thousand classes, the number of classes were limited only to thousand to match its performances.

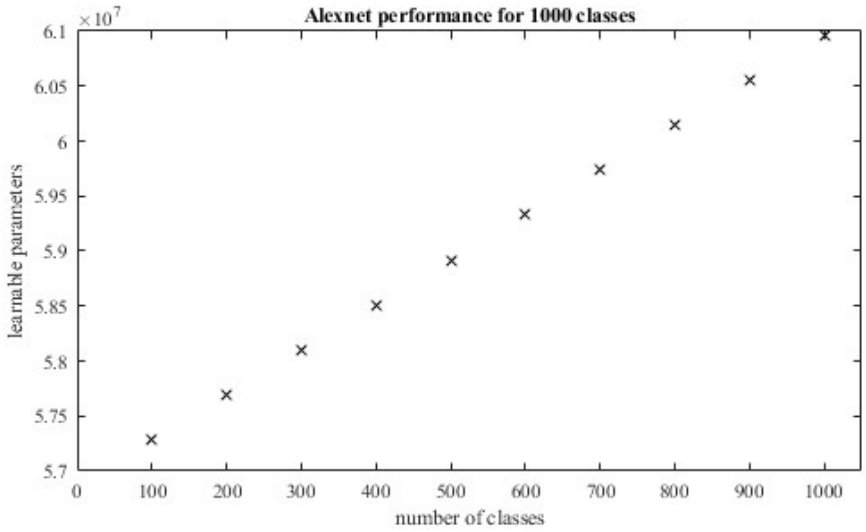


Figure 4.8: AlexNet network performance for number of classes from 100 to 1000

Since the combination of classes for both main and secondary networks are selected, out of all possible value combinations, the minimum value is selected. Those values are tabulated in appendix A along with equivalent data for the re-trained AlexNet network. Comparing the learnable parameters calculated for the AlexNet network and cascaded neural network model, based on the plots presented above, it can be stated that proposed cascaded convolutional neural network is 6 times faster than the conventional AlexNet network. Therefore, it can be proved that cascading convolutional neural network with minimum computation involved can produce faster inferences compared to deep convolutional neural network models.

4.5 Classification accuracies

It is equally important to present the classification accuracies of the proposed cascaded neural network model in addition to its execution speed performances, as it provides more reliability measurement to the output.

The classification accuracy of any convolutional neural network is a great measure of how accurately the network model can predict class label of the given input object of interest out of all the objects which is learned to identify. Classification accuracy can be denoted as a mathematical formula in Equation 4.1 below.

$$\text{Classification Accuracy} = \left(\frac{\text{Number of correct predictions}}{\text{Total number of predictions}}\right) \times 100 \quad (4.1)$$

The proposed network model was evaluated for its classification accuracy to determine its performance as compared to other existing network models. Commonly used error matrices were employed for the accuracy analysis. Error distance (ED_i) is the quantitative measure which represents the arithmetic difference between the approximated result ($\acute{R}_i$) and the actual result (R_i) mathematically represented in Equation 4.2.

$$\text{ED}_i = \left|\acute{R}_i - R_i\right| \quad (4.2)$$

Mean Error Distance (Mean error distance (MED)) is the average error distance for N number of test trials can be mathematically represented in Equation 4.3.

$$\text{MED} = \frac{1}{N}\sum_{i=1}^{N}\text{ED}_i \quad (4.3)$$

Considering the relative error distance $(RED)_i$), which is a measure of error distance (ED) with respect to the actual result (R) how much deviated from the actual result. Relative error distance can be mathematically

denoted in Equation 4.4.

$$\mathrm{RED}_i = \left| \frac{\acute{R}_i - R_i}{R_i} \right| \tag{4.4}$$

Mean Relative Error Distance ($Meanrelativeerrordistance(MRED)$) is the average relative error distance for N number of test trials which can be mathematically represented in Equation 4.5.

$$\mathrm{MRED} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{RED}_i \tag{4.5}$$

Mean Square Error (MSE) is the average of the squared error distances, which can be mathematically denoted in Equation 4.6.

$$\mathrm{MSE} = \frac{1}{N} \sum_{i=1}^{N} |\mathrm{ED}_i|^2 \tag{4.6}$$

To ensure a rigorous and scientifically valid comparison between the proposed cascaded CNN and other state-of-the-art models, it is essential to quantify the uncertainty in the measured accuracy and inference time. Machine learning results are affected by sampling variation, and the reported performance metrics reflect only a single realization of an underlying probability distribution. This section discusses the confidence intervals for classification accuracies and inference times. Classification accuracy is defined as the proportion of correctly classified samples in a test set. Let k denote the number of correct predictions among N total test samples. The observed accuracy is denoted in Equation 4.7.

$$\hat{p} = \frac{k}{N} \tag{4.7}$$

Since accuracy is a proportion which follows binomial distribution, for sufficiently large samples, the accuracy $\hat{p}$ is the sample mean represented in the Equation 4.8.

$$\hat{p} = \frac{1}{N} \sum_{i=1}^{N} X_i \tag{4.8}$$

According to the central limit theory for a reasonably large samples N, the distribution of sample mean will become equal to standard error. Since each prediction corresponds to a Bernoulli trial (correct or incorrect),accuracy follows a binomial distribution. Considering Bernoulli variable, mean and variance is represented in Equation 4.9 and 4.10.

$$E[X_i] = \hat{p} \tag{4.9}$$

$$Var(X_i) = \hat{p}(1 - \hat{p}) \tag{4.10}$$

Therefore, adapting central limit theory for large samples N the standard error of a binomial proportion is denoted in Equation 4.11.

$$SE = \sqrt{\frac{\hat{p}(1 - \hat{p})}{N}}. \tag{4.11}$$

For a normal approximation, a 95% Wald confidence interval for the true accuracy $\hat{p}$ is denoted by Equation 4.12.

$$CI_wald = \hat{p} \pm 1.96 \tag{4.12}$$

Substituting into the standard error equation, the Wald confidence interval for 95% accuracy is represented by Equation 4.13.

$$CI_{wald95\%} = \hat{p} \pm 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{N}}. \tag{4.13}$$

The Wald classification confidence interval is most effective with large samples. However, it tends to perform poorly with small samples, as its accuracy is often close to 0 or 1. This overconfidence can be particularly evident in CNN tests. Therefore, for small samples, the Wilson confidence interval is recommended. Let z be the standard normal critical value ($z = 1.96$ for 95%confidence). The adjusted proportion in the Wilson method is indicated in Equation 4.14.

$$\hat{p}_{Wilson} = \frac{\hat{p} + \frac{z^2}{2N}}{1 + \frac{z^2}{N}}. \tag{4.14}$$

The corresponding Wilson margin of error is represented in Equation 4.15.

$$ME_{Wilson} = \frac{z}{1 + \frac{z^2}{N}}\sqrt{\frac{\hat{p}(1-\hat{p})}{N} + \frac{z^2}{4N^2}}. \tag{4.15}$$

Therefore, Wilson confidence interval is denoted in the Equation 4.16.

$$CI_{\text{Wilson}} = \hat{p}W \pm ME_Wilson. \tag{4.16}$$

By substituting Wilson center and margin, the Wilson confidence interval can be expressed using the following Equation 4.17.

$$CI_{\text{wilson}} = \frac{\hat{p} + \frac{z^2}{2N}}{1 + \frac{z^2}{N}} . \frac{z}{1 + \frac{z^2}{N}}\sqrt{\frac{\hat{p}(1-\hat{p})}{N} + \frac{z^2}{4N^2}}. \tag{4.17}$$

Additionally, two-proportion Z test can be used to compare the accuracies of two models.Let $\hat{p_1}$, $\hat{p_2}$ represent the observed accuracies of two models

which can be denoted by the Equation 4.18.

$$\hat{p}1 = \frac{k1}{N_1}, \qquad \hat{p}2 = \frac{k2}{N_2} \tag{4.18}$$

Two proportion Z-test is represented by Equation 4.19.

$$Z = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}(1-\hat{p})\left(\frac{1}{N_1} + \frac{1}{N_2}\right)}} \tag{4.19}$$

Similarly, the confidence interval for the inference time can be tested over multiple runs.A confidence interval (CI) provides a range within which the true mean inference latency lies with a specified level of confidence.

Let $t_1, t_2, \ldots, t_n$ be n measured inference times (in milliseconds) of a CNN model. The average inference time $\bar{t}$ is denoted by Equation 4.20.

$$\bar{t} = \frac{1}{n}\sum_{i=1}^{n} t_i. \tag{4.20}$$

The standard deviation across n runs can be obtained from Equation 4.21.

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(t_i - \bar{t})^2}. \tag{4.21}$$

The margin of Error can be derived from the t-distribution represented in Equation 4.22.

$$\text{ME} = t_{\alpha/2,\, n-1} \cdot \frac{s}{\sqrt{n}}. \tag{4.22}$$

Therefore, Confidence interval for inference time is represented in the Equation 4.23.

$$\text{CI} = \bar{t} \pm \text{ME}. \tag{4.23}$$

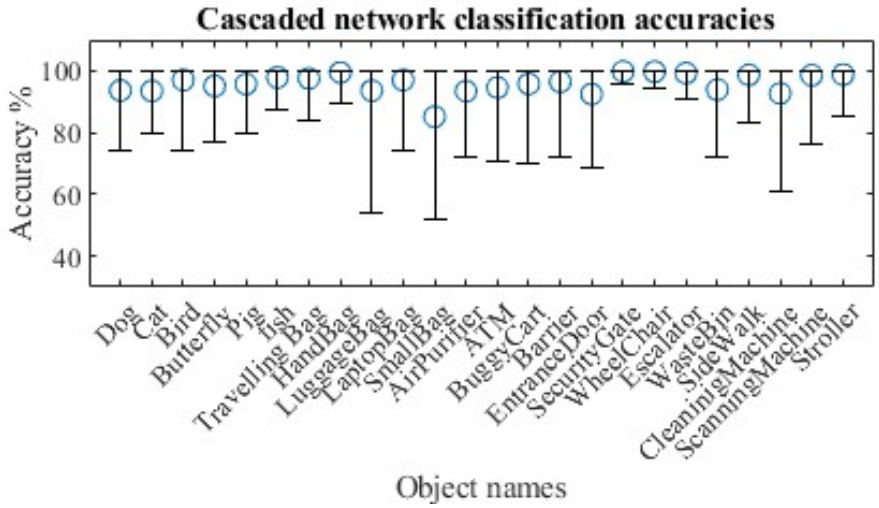


Figure 4.9: Cascaded network classification accuracies

Since the classification performance of the cascaded network should be compared with an existing network model, the pre-trained Alex net network was chosen. The Alex net network model trained over one thousand object classes was re-trained with the dataset created for the cascaded neural network model using transfer learning. The dataset created for the cascaded network model consists of a large number of sample images for pre-configured object classes. The experiment was conducted on the same computer used for the previous performance analysis. For each category, the sample images were fed through both cascaded network as well as the re-trained Alex net. The results were tabulated and presented in Figure 4.9.

Figure 4.9 presents the classification accuracies achieved by the cascaded network model on different inputs of images. The error bars indicate the range of accuracy values (maximum and minimum) recorded for each class. As shown, the model achieves an overall classification accuracy of 95%, with the lowest class accuracy at 75%. Although the model generally delivers strong performance, one class exhibited a lower average accuracy of 93% and the lowest confidence score of 54% among all test classes. Similarly,

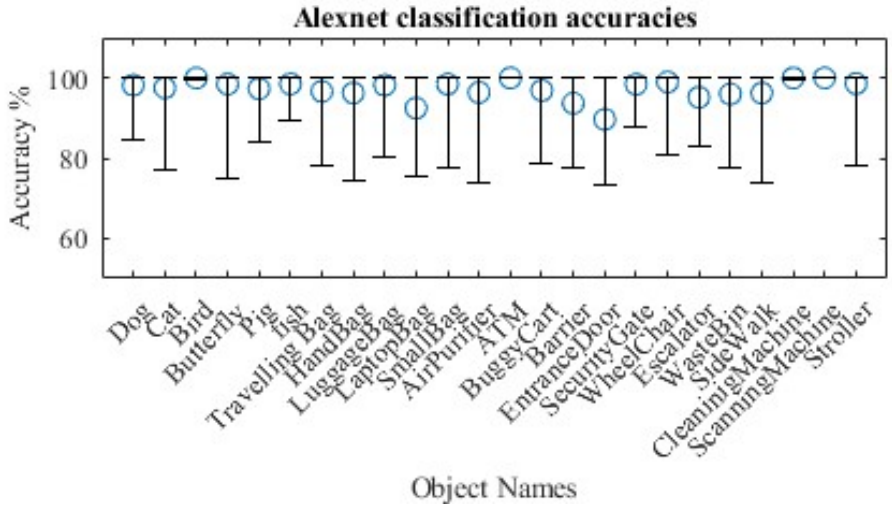


Figure 4.10: Alex net re-trained network classification accuracy

the re-trained AlexNet from the transfer learning was tested on the same object categories using the same test images prepared to be tested over the proposed network model. The results were tabulated and presented in Figure 4.10.

As shown in Figure 4.10, the proposed cascaded network model achieved an average classification accuracy of 97%, with the lowest accuracy across object classes at 83%. Notably, even the least accurate class (75%) still outperformed most baseline models. Compared to the AlexNet model, the proposed model's accuracy is only 2% lower, yet it delivers predictions in the shortest inference time. Overall, the model demonstrates satisfactory accuracy with superior computational efficiency. For testing purposes, this research focused on classification accuracy and inference time. However, confusion matrices were not adapted, as they are more suitable for evaluating class-wise discrimination in large labeled datasets, particularly when compared to fully supervised models. The proposed architecture emphasizes real-time operational behavior rather than static classification on large datasets. Manual testing allowed for precise control over test conditions, such as object placement, lighting, and viewpoint, which are often

difficult to standardize in static benchmark datasets. This approach more accurately reflects realistic deployment scenarios than offline test datasets.

Misclassification cases, where the cascaded CNN under-performs compared to AlexNet, are both expected and acceptable. This is because the proposed architecture is optimized for real-time, edge-based operation with a focus on robustness and efficiency rather than achieving peak accuracy alone. Misclassifications may occur due to the shallower and pruned first stage of the cascaded network, which is designed for speed. While AlexNet may correctly classify a misclassified image from the cascaded network, it does so with low confidence. If Monte Carlo dropout was applied during the inference stage, these poor classification cases could be clearly identified. However, it is important to note that this under-performance is not an indication of failure; rather, it highlights areas that can be refined in the secondary stage.

The proposed cascaded architecture is designed to enhance generalization reliability by distinguishing between confident predictions and uncertain cases through a multi-stage decision mechanism. Unlike state-of-the-art single-stage networks like AlexNet, which must provide a classification for every input regardless of confidence, the cascaded model selectively advances uncertain samples to a deeper refinement stage. For noisy or previously unseen inputs, the first-stage network is expected to show reduced confidence or increased predictive entropy, which triggers the escalation to the second-stage network for further processing. This approach reduces the likelihood of confident misclassification and improves the decision reliability in the presence of domain shifts, occlusion, blur, illumination variation, and sensor noise. In contrast, AlexNet lacks an uncertainty-aware fallback mechanism; therefore, it cannot adapt its computational depth based on the difficulty of the input, making it more susceptible to misclassification

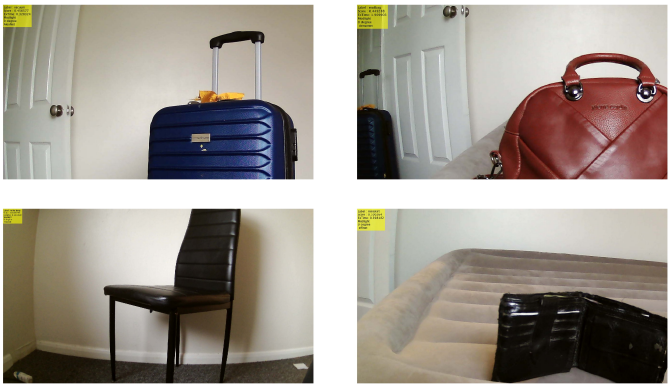


Figure 4.11: Classification results for real-time images of test objects

under degraded conditions.

In addition to the test dataset used for model evaluation, real-time images were utilized to assess the classification capabilities of both the proposed models and state-of-the-art models. A USB camera connected to the computer captured real-time images, which were then used as input for the CNN model. Sample results from various CNN models tested on randomly selected objects are presented in Figure 4.11.

The MATLAB remote debugging tool was used to test various convolutional neural network (CNN) models, including the proposed cascaded network, on a Raspberry Pi computer. A USB camera connected to the Raspberry Pi captured real-time images of test objects and executed CNN models on board. The sample results are presented in Figure 4.12. The MATLAB code generation tool was used to construct equivalent C++ code from the scripts written to run on the Raspberry Pi.

The proposed cascaded network was executed on a Raspberry Pi A400 computer. To perform real-time image classification, a USB camera was attached to the A400 for capturing images for model evaluation. The main network and the sub-network were executed sequentially. Sample test results are presented in Figure 4.13 . The classification results clearly indi-

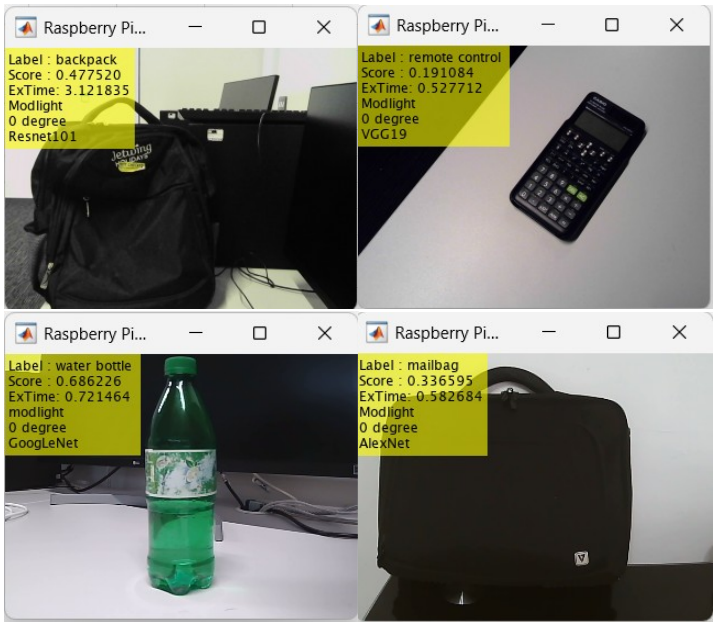


Figure 4.12: Classification results for real-time images of test objects executed on Raspberry Pi

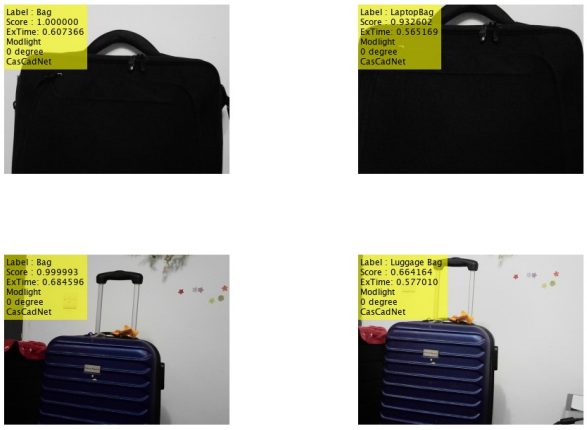


Figure 4.13: Cascaded network classification results for real-time images of test objects executed on a Raspberry Pi A400 computer.

cate that the sub-network refines the output of the main network. This also highlights the potential for early exits, which can provide improved classification confidence while saving execution time and increasing accuracy.

4.6 CNN Model Benchmarking

This section presents the benchmarking results of various Convolutional Neural Network (CNN) architectures used for image classification of target objects commonly found in an airport environment. Real-time test images were captured using a USB camera connected to a computer. The experiment aims to evaluate and compare the performance of nine pre-trained CNN models: AlexNet, GoogLeNet, ResNet50, ResNet101, VGG16, VGG19, DenseNet201, and EfficientNetB0, alongside two custom-developed architectures called MainNet and SubNet. The benchmarking focuses on three main performance indicators:

1. Classification Confidence (%): This measures the likelihood that a model correctly assigns an object to its predicted class.

2. Inference Time (seconds): This refers to the amount of computational time required to process a single image.

3. Prediction Accuracy: This indicates how closely the predicted class label aligns with the actual object.

All experiments were conducted in two computing environments: a standard computer equipped with an NVIDIA RTX 3050 GPU and a Raspberry Pi A400. The dataset consisted of real-world images of various test objects, such as bags and furniture, all captured under consistent lighting and background conditions. Each CNN model was tested on various object classes to assess its ability to generalize and accurately classify visually similar categories, such as handbags, luggage, and furniture. The average results across all test cases are summarized in Table 4.11.

Table 4.11: Model Evaluation Summary of CNN Architectures using real-time data

Model	**Average Confidence (%)**	**Average Inference Time (s)**	**Overall Accuracy**
AlexNet	39.5	0.27	Low
GoogLeNet	47.1	0.43	Medium
ResNet50	47.7	1.34	Medium
ResNet101	56.7	2.33	Medium
VGG16	47.4	0.37	Medium
VGG19	43.2	0.41	Medium
DenseNet201	61.5	2.07	High
EfficientNetB0	34.4	0.68	Medium
MainNet	96.9	0.25	High
SubNet	83.9	0.21	High

Classification confidence:

The classification confidence and inference time of each state-of-the-art model are presented in a bar chart in Figure 4.14. As illustrated in this figure, the custom models significantly outperform standard CNN architectures in terms of classification confidence. The MainNet achieved the highest mean confidence of 96.9%, followed by SubNet at 83.9%. Among the standard networks, DenseNet201 demonstrated the strongest performance with a confidence level of 61.5%, while EfficientNetB0 recorded the lowest at 34.4%.

State-of-the-art architectures like AlexNet and GoogLeNet exhibited lower confidence due to their shallow layer structures and limited feature extraction capabilities. Meanwhile, modern deep networks such as ResNet and VGG achieved moderate confidence but were computationally slower.

These results suggest that the MainNet's deeper yet optimized structure facilitates more robust feature extraction, allowing for better generalization across visually similar categories, such as handbags and laptop bags.

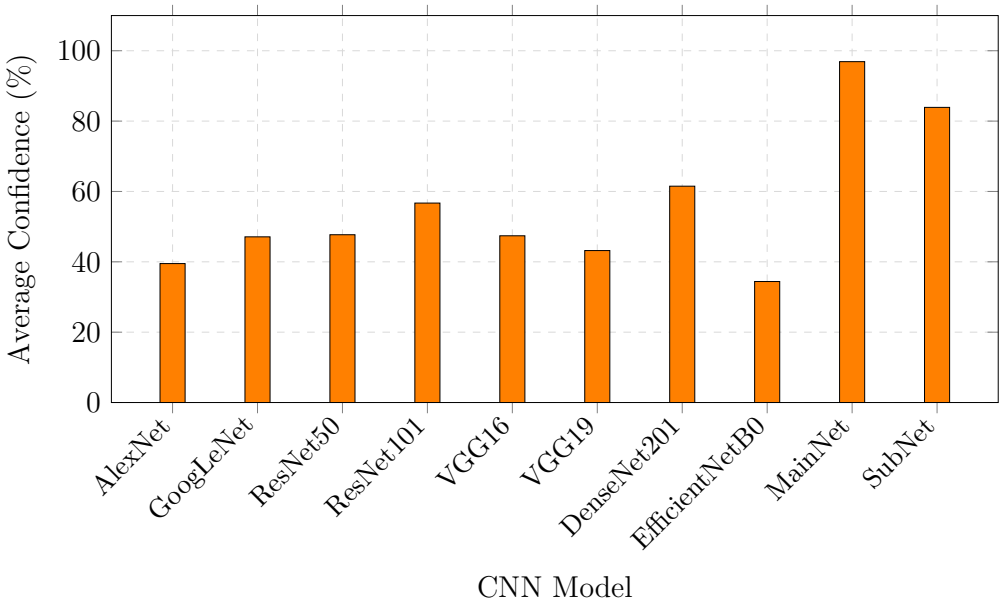


Figure 4.14: Average Confidence by CNN Model

Inference results:

The inference time analysis (Figure 4.15) shows that SubNet achieved the fastest average inference time of 0.21 seconds per image, closely followed by MainNet at 0.25 seconds. In contrast, ResNet101 and DenseNet201 were the slowest models, averaging 2.33 seconds and 2.07 seconds per image, respectively. This highlights the computational trade-off between deeper architectures and efficiency.

The results indicate that MainNet and SubNet strike a favorable balance between speed and accuracy, making them suitable for real-time applications such as embedded vision systems, smart surveillance, and autonomous inspection.

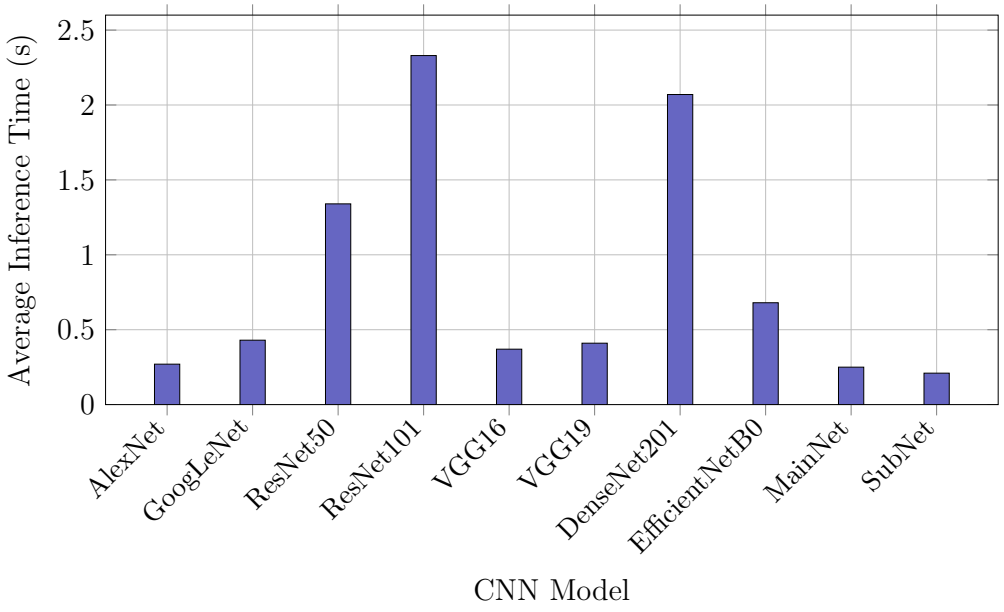


Figure 4.15: Average Inference Time by CNN Model

Comparative performance:

Figure 4.16 presents a combined comparison of model confidence and inference time. The chart clearly distinguishes the performance between standard CNNs and the custom architectures.

While pre-trained state-of-the-art models struggled with high intra-class similarity such as misclassify handbags as backpacks or wallets as purses—the MainNet and SubNet networks displayed superior adaptability. This improvement suggests that their training included more discriminative feature representations specific to the object domain.

Additionally, MainNet achieved a higher classification accuracy compared to its counterparts, demonstrating a strong generalization capability across a broader range of classes by extracting features with minimal inference delay. This indicates the potential for early exits in the classification process when a higher confidence score is reached. Although SubNet was slightly less accurate, it remained the fastest model overall. It can be viewed as a further refinement of the earlier stages, suggesting a more concise and

human-like approach to artificial intelligence.

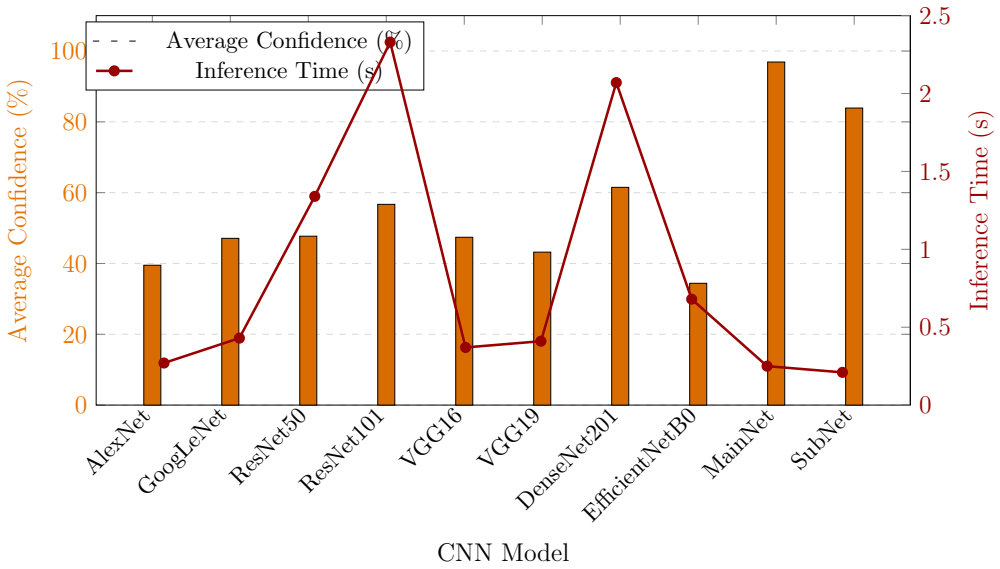


Figure 4.16: Combined comparison of Average Confidence and Inference Time across CNN models, showing dual y-axes with unified legend below the plot.

The experimental findings confirm that custom CNN architectures designed specifically for the dataset significantly outperform general-purpose, pre-trained models.

Several key observations emerge:

1. Performance-Accuracy Trade-off: Traditional models (e.g., ResNet, VGG, DenseNet) require considerably higher inference times to achieve moderate accuracy. In contrast, MainNet achieves superior results with lower computational costs due to its architectural optimization and parameter efficiency.

2. Feature Specialization: MainNet and SubNet have effectively adapted to the dataset, allowing for accurate differentiation among similar-looking objects. This specialization is a result of fine-tuning and possibly enhanced attention mechanisms within their convolutional layers.

3. Scalability and Real-Time Application: With their high confidence and low inference time, MainNet and SubNet are well-suited for deployment in real-time image recognition environments, such as embedded camera systems or mobile platforms.

4. Misclassifications Trends: Standard CNNs often confuse semantically similar categories (e.g., handbags versus laptop bags). Custom networks significantly reduce these errors, demonstrating improved feature abstraction for texture and shape-based classification.

In conclusion, the benchmarking study shows that MainNet and SubNet outperform standard state-of-the-art CNN architectures in both classification confidence and inference efficiency.

The real-time experiment revealed that MainNet achieved an overall accuracy of 96.9% with an inference time of 0.25 seconds, while SubNet reached a confidence level of 83.9% with an inference time of 0.21 seconds. These findings underscore the effectiveness of customized architectures for domain-specific image recognition tasks and highlight their suitability for real-time applications that demand both precision and computational efficiency.

4.7 Chapter Summary

This chapter provides a thorough analysis of the experimental results derived from the implementation and evaluation of the proposed cascaded convolutional neural network (CNN) architecture. The findings confirmed that the cascaded design effectively balances accuracy and computational efficiency, addressing the trade-off challenges typically encountered in deep

learning applications on embedded and edge platforms. The primary CNN delivered rapid inference for most input images, while the secondary CNN successfully refined low confidence predictions, resulting in a consistent improvement in overall classification accuracy. Comparative evaluations with baseline models like AlexNet and MobileNet showed a significant improvement in performance, achieving speeds approximately six times faster than state-of-the-art models while maintaining the same classification accuracy during execution. These enhancements validate the effectiveness of the architectural optimizations implemented, which include efficient layer design, pruning, hardware-aware computation, and layer fusion techniques. Additionally, the experimental results support the research hypothesis that a cascaded CNN structure can deliver real-time classification performance while maintaining high accuracy in constrained computational environments. This chapter also demonstrated the scalability of the proposed model, confirming its suitability for real-world applications such as autonomous guidance systems and service robots that operate on low power edge devices. The following chapter will present the conclusions and future research directions, summarizing the key findings of this work and outlining potential improvements to further optimize CNN performance for energy-efficient, real-time embedded intelligence.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

Contents

5.1 Main contributions **188**

5.2 Model effectiveness **192**

5.3 Research challenges and dataset limitations **193**

5.4 Summary of chapters **195**

5.5 Future work **196**

5.5.1 Multistage cascaded network 196

5.5.2 Multi-feature extraction architecture 197

Bibliography **198**

5.1 Main contributions

This section presents an overview of the contribution of the work and its achievements. The research work is focused on enhancement of inference

speed of a neural network by proposing a novel network architecture. The proposed design consists of two convolutional neural network models with the least number of computational parameters. The architectural design was inspired by the most successful neural network models after a deep analysis of signature characteristics. The cascaded network model was tested on a large dataset. Enhanced performances were observed from the cascaded network architecture. The development of main network architecture was concerned on reduction of unnecessary computations involved in learnable layers such as convolutional and fully connected layers and thereby minimize the inference time. Filter kernels used in convolutional stages were selected for optimized operation ensuring minimum losses during the learning period. Inspired by the inception module a branch connection of convolutional layers with small kernel sizes were adapted towards the end of the main network architecture to enhance the generalization capability. Since the branch was executed concurrently reduced the execution time. Main network model was trained for fewer number of classes with more data samples compared to the subnetwork which was trained over broader classes. The subnetwork was designed to meet the criteria of minimum computations involved to ensure that minimum execution time offered from the cascaded network model. The subnetwork architecture was trained over a greater number of classes which offered more specific classification ability of individual classes in less time as compared with other network architectures. From the results achieved, the cascaded network offered reduced execution time as compared with Alex Net over the increasing number of output classes. The proposed cascaded network model is 6.43% faster than the standard AlexNet model with the same number of output classes. Comparing classification accuracies, the cascaded network model is 2% less confident to the equivalent of standard AlexNet. However, the accuracy of the cascaded model can be further improved

with the suggestions presented in subsection 5.3.1 and 5.3.2. Another important finding was presented where the relationship between number of output classes and the learnable parameters where lesser number of classes in main network and a greater number of classes in subnetwork results in optimum combination offering least execution time. This result could be linked to achieving an efficient learning model which represents how effectively the knowledge must be separated for a fast and accurate response to achieve. As an example, identification of a physical object should be first explained with geometric appearance following a color, contour, and textural estimation. Identification of geometric appearance is equivalent to categorization in main network and subsequent classification equivalent to sub network. The experiments were initially conducted on one computer to acquire results. The network models were trained with dataset created from the CIFAR-10 dataset originally used by Krizhevsky et al in AlexNet. The testing of the trained models was conducted on various computers with different CPU configurations and additional computations such as GPUs. It was observed that the execution time measured depends on various factors which are linked to the computation process such as CPU type, cache memory, availability of additional computational resources such as GPU etc. The test was conducted on different computers with hardware configurations such as different processor types, cache memories, random access memories (RAM)s with variable bus speeds. However, the execution time of the network model was dependent on the computational configurations, the classification accuracy remained unchanged irrespective of the hardware resources. Although cascaded architectures initially used for fast and accurate semantic segmentation of multiclass object detection, its proven results on fast execution motivated to adapt cascaded architecture to the computationally optimized CNN models for image classification in more pragmatic way with better generalization capacity and fast processing. The classifi-

cation demonstrated was more realistic rather straight forward segregation into classes based on convergence of neurons. The primary contribution of this research was to reduce the complex computations involved in convolutional neural networks. Modifications introduced to the neural network architecture in terms of layer wise changes have been a significant factor for reduction of unnecessary computations as compared with standard neural models. As a result of the modifications introduced to the network model, a considerably large reduction of learnable parameters was observed which reflects the effective computation optimizations compared with other network models such as GoogLeNet. Architectural modifications introduced with this research is the second contribution which includes convolution with varying kernel sizes from, batch normalizations, concurrent feature extractions etc. Due to the modifications introduced to the architecture resulted in minimalizing of computations and comparisons in both sparse and dense layers. Since the number of complex computations are reduced, the time taken to execute the network also reduced and thus the inference time. Parallel feature extraction towards the last convolutional layers has significantly reduced the execution time of the network model and thereby speeding up the classification process. Adaptation of computationally efficient functions such as rectified and linear units for the feature activation instead of parametric rectified linear unit, tanh, or exponential linear units. It not only overcomes unnecessary computations but also minimizes the time to compute exponents. The third contribution of the work is the proposed network model maintains better classification accuracy while optimizing the computations and reducing inference time. Normalization techniques introduced after every feature activation has resulted in better classification capability of the proposed network model. The contributions mentioned above successfully fulfill the objectives mentioned in section 1.3 and extensions of the work are presented in section 5.3.

5.2 Model effectiveness

This research conducted a comprehensive benchmarking study to evaluate the performance of the proposed cascaded CNN architectures against widely adopted state-of-the-art deep learning models, including AlexNet, GoogLeNet, ResNet50, ResNet101, VGG16, VGG19, DenseNet201, and EfficientNetB0. The evaluation utilized real-time image data captured from a camera module attached to a computer, focusing on objects in an airport-like environment. Three core performance indicators were assessed: classification confidence, inference latency, and overall prediction accuracy. The experimental results demonstrate that the proposed two stage cascaded architecture, named MainNet and SubNet, consistently outperforms generic pre-trained models across all evaluated metrics. Specifically, MainNet achieved the highest overall classification confidence of 96.9% while maintaining a low inference time of just 0.25 seconds, offering an excellent balance between accuracy and computational efficiency. SubNet further enhanced real-time performance by achieving the lowest average inference time of 0.21 seconds while still maintaining a high confidence level of 83.9%. In contrast, deeper and more complex architectures such as ResNet101 and DenseNet201 required significantly longer inference times, exceeding two seconds per image, without providing comparable improvements in classification confidence. This highlights the significance of the architectural depth when applied to domain specific real-time classification tasks, where excessive model complexity introduces computational overhead without proportional performance gains. It also highlighted the possibility for early exits for satisfactory performance from the prime stage. Saving further computation resources further improving the effectiveness of the proposed architecture. A key observation from the benchmarking analysis is

the advantage of feature specialization in the proposed cascaded networks. While standard CNN models frequently misclassified semantically similar objects, such as handbags and laptop bags, the custom architectures demonstrated a markedly improved ability to discriminate between visually similar categories. This suggests that the architectural design enabled for more discriminative and task relevant feature representations. Overall, the benchmarking results validate the central hypothesis of this research, that the lightweight domain specific CNN architectures offer higher practical performance than general purpose deep learning models in real-time edge environments such as Raspberry pi. The proposed cascaded approach effectively balances accuracy, latency, and model complexity, making it particularly suitable for embedded vision based systems, where both responsiveness and reliability are critical.

5.3 Research challenges and dataset limitations

The proposed cascaded CNN framework showed strong performance in various experimental scenarios. However, several practical challenges and limitations were encountered during this research. We acknowledge these factors to provide a transparent and balanced interpretation of the results and to highlight opportunities for future improvement.

One of the main challenges we faced was the availability and representativeness of the dataset. The experimental dataset was created using real-time images captured under controlled environmental conditions, along with selected samples from the ImageNet ILSVRC dataset. While this approach allowed for reproducible benchmarking and consistent evaluation across

various CNN architectures, the resulting dataset does not fully represent the variability found in real-world situations. This includes issues such as extreme lighting changes, motion blur, heavy occlusions, and diverse camera viewpoints. As a result, although the proposed models demonstrated strong generalization within the evaluated domain, their performance in highly dynamic or adversarial conditions remains an area that requires further investigation. One limitation of the study is related to the scale and diversity of the dataset. Although it incorporates multiple object categories relevant to service robots and surveillance applications, the size of the dataset was limited by practical factors such as data collection, the effort required for annotation, and available computational resources. Consequently, some object classes were underrepresented, which may lead to class imbalance and reduce the system's robustness in rare or edge case scenarios. Additionally, hardware limitations restricted large scale experimentation. While testing was performed on a workstation with a GPU and on an edge device, more extensive cross platform evaluations across different types of hardware, such as mobile GPUs or specialized accelerators, were not included in this study. Conducting such evaluations could yield valuable insights into scalability and deployment feasibility.

Despite these limitations, the research successfully met its primary objectives by demonstrating that well designed, lightweight convolutional neural network (CNN) architectures can outperform general purpose deep learning models in domain specific real-time environments. The challenges identified highlight the practical constraints encountered in applied deep learning research and provide valuable guidance for future work. This is particularly relevant in areas such as dataset expansion, robustness evaluation under noisy conditions, and the development of uncertainty aware decision making mechanisms.

5.4 Summary of chapters

Chapter1 presented an overview of the research, with a preliminary background related to artificial neural networks. A brief explanation on convolutional neural networks, their current state, and the limitations presented, followed by the motivation to conduct this research. The research presents a clear problem statement, outlining the objectives and milestones achieved.A summary of the original research contribution was included at the end of the section. Chapter 2 introduced a background study with convolutional neural networks. Beginning with the very first perceptron model, up to the modern deep neural network models were explained with the key features which had led to its success. The key constraints upon which one decides the performance of a convolutional neural network are described in this section. This proposes a tangible solution for improving the convolutional neural network model. Chapter 3 the prior knowledge gathered with a clear road map, the proposed neural network model, the cascaded convolutional neural network thus presented in stages with necessary mathematical support. The theoretical explanation of the novel network model was built and tested on a computer where the networks trained on large dataset. Chapter 4 the results obtained from the proposed network model were carefully analyzed. The performance of the proposed network model was compared with one of the most cited network architectures. The results obtained were used to support the argument that inference time of a convolutional neural network can be minimized by cascading neural network model with minimum learnable parameters. The results obtained were contrasted with the performance of existing network architecture.

5.5 Future work

The potential future research can also be presented based on the developments of the research work. Further analysis of the results obtained in section 4, a tendency of minimum learnable parameters can be observed for the selection of number of classes to the main and sub network. The potential combinations for the classes were centered to minimum values possible. Thus, cascading of third and higher order convolutional neural networks are a potential feature where the present two stages can further be extended. Development of hierarchical dataset is essential to facilitate more meaningful and human-like classification. Further development of this cascaded approach has the potential to deploy on mobile robotic systems with standalone computer platforms which work offline and function efficiently. However, a deep analysis of the network is essentially supported with mathematical representation.

5.5.1 Multistage cascaded network

The proposed model consists of two stage cascaded network for classification applications with lesser number of classes. This concept can be further expanded by cascading a greater number of optimized networks together for a large number of classes. The optimum number of cascades can be determined by conducting practical execution time measurement over the number of classes. Future work is required to investigate the performance of multi stage cascaded neural network.

5.5.2 Multi-feature extraction architecture

The neural network model presented with this research work consists of modified Alex net architecture. Convolution operation of an intermediate layer depends on feature map developed from the previous layer convolved with input feature map followed by an activation function. Therefore, convolution operation towards the dense layers is dependent on the features extracted from prior layers. Instead of single feature extraction, if multiple features could be extracted concurrently and concatenate them at the last convolutional layers possibly improve classification accuracy in considerably less time. Further investigation of the architectural modification with deep training and analysis of results can result in more accurate and efficient models which can be cascaded.

Bibliography

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov. 1998.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, pp. 84–90, May 2012.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Boston, MA, USA), pp. 1–9, IEEE, 2015.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Chapter 9: Convolutional Networks*, pp. 326–366. Cambridge, Massachusetts: The MIT Press, 2017.

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, pp. 541–551, Dec. 1989.

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[7] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), vol. 8689 of *Lecture Notes in Computer Science*, (Cham), pp. 818–833, Springer International Publishing, 2014.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Las Vegas, NV, USA), pp. 770–778, IEEE, 2016.

[9] M. Lin, Q. Chen, and S. Yan, "Network in network." arXiv preprint arXiv:1312.4400, 2013. Available: http://arxiv.org/abs/1312.4400.

[10] B. Siegel, "Industrial anomaly detection: A comparison of unsupervised neural network architectures," *IEEE Sensors Letters*, vol. 4, Aug. 2020.

[11] K. D. Rusakov, "Automatic modular license plate recognition system using fast convolutional neural networks," in *2020 13th International Conference "Management of large-scale system development" (MLSD)*, (Moscow, Russia), pp. 1–4, IEEE, 2020.

[12] S. Gautam and J. Singhai, "Fast and accurate water region classification from remote sensing images using enhanced convolutional neural network classifier," in *2022 IEEE International Students' Conference*

on Electrical, Electronics and Computer Science (SCEECS), (Bhopal, India), pp. 1–4, IEEE, 2022.

[13] J. Lee, Y. An, M. Kim, and S. Pan, "Comparison of cnn architecture for thermal face emotion classification," in *2021 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, IEEE, 2021.

[14] Y. Liu, M. Xiang, X. Zhao, and R. Zhou, "Design of fast image recognition accelerator based on convolutional neural network," in *2020 4th Annual International Conference on Data Science and Business Analytics (ICDSBA)*, (Changsha, China), pp. 304–307, IEEE, 2020.

[15] S. Zhai, S. Dong, D. Shang, and S. Wang, "An improved faster r-cnn pedestrian detection algorithm based on feature fusion and context analysis," *IEEE Access*, vol. 8, pp. 138117–138128, 2020.

[16] J. Huang, S. Huang, Y. Zeng, H. Chen, S. Chang, and Y. Zhang, "Hierarchical digital modulation classification using cascaded convolutional neural network," *Journal of Communications and Information Networks*, vol. 6, pp. 72–81, Mar. 2021.

[17] Y. Luo, C. Wu, and C. Lv, "Cascaded convolutional recurrent neural networks for eeg emotion recognition based on temporal–frequency–spatial features," *Applied Sciences (Switzerland)*, vol. 13, June 2023.

[18] S. Huang and W. Yu, "Cascade prediction with recurrent neural networks and diffusion depth distributions," in *2023 3rd International Conference on Neural Networks, Information and Communication Engineering (NNICE)*, pp. 70–77, IEEE, 2023.

[19] S.-Y. Sun, H. Xu, J. Li, H. Liu, and Q. Li, "Cascaded neural network for memristor based neuromorphic computing," in *2019 Inter-*

national Joint Conference on Neural Networks (IJCNN), (Budapest, Hungary), pp. 1–6, IEEE, 2019.

[20] M. W. Hirsch, "Convergence in cascades of neural networks," in *International 1989 Joint Conference on Neural Networks*, vol. 1, (Washington, DC, USA), pp. 207–208, 1989.

[21] Y. Ren and B. Wang, "Fast human detection via a cascade of neural network classifiers," in *IET 3rd International Conference on Wireless, Mobile and Multimedia Networks (ICWMNN 2010)*, (Beijing), pp. 323–326, 2010.

[22] T. Shen and D. Zhu, "Layered_casper: Layered cascade artificial neural networks," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, (Brisbane, QLD, Australia), pp. 1–7, 2012.

[23] A. J. Dhanaseely, S. Himavathi, and E. Srinivasan, "Performance comparison of cascade and feed forward neural network for face recognition system," in *International Conference on Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012)*, 2012.

[24] Z. Lu and C. Tang, "Research on grouping-cascaded bp network model," in *2010 2nd International Conference on Advanced Computer Control*, (Shenyang, China), pp. 425–429, 2010.

[25] S. Durvasula, Y. Guan, and N. Vijaykumar, "Ev-conv: Fast cnn inference on event camera inputs for high-speed robot perception," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3174–3181, 2023.

[26] T. M. Phuong and N. N. Diep, "Speeding up convolutional object detection for traffic surveillance videos," in *2018 10th International*

Conference on Knowledge and Systems Engineering (KSE), (Ho Chi Minh City, Vietnam), pp. 13–18, 2018.

[27] K. Fukushima, "Neocognitron: A Self-Organizing Neural Network for a Mechanism of Pattern Recognition Unaffected by Shift in Position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[28] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525, IEEE, Nov. 2017.

[29] Y. Nie, P. Sommella, M. O'Nils, C. Liguori, and J. Lundgren, "Automatic Detection of Melanoma with YOLO Deep Convolutional Neural Networks," in *2019 E-Health and Bioengineering Conference (EHB)*, (Iasi, Romania), pp. 1–4, IEEE, 2019.

[30] A. Lavin and S. Gray, "Fast Algorithms for Convolutional Neural Networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2016.

[31] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast Convolutional Nets With fbfft: A GPU Performance Evaluation," 2014. Preprint.

[32] S. Hong and D. Park, "Continuous differential image-based fast convolution for convolutional neural networks," in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, (Jeju Island, Korea), pp. 492–494, 2022.

[33] J. Lee, Y. An, M. Kim, and S. Pan, "Comparison of cnn architecture for thermal face emotion classification," in *2021 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, (Gangwon, Korea), pp. 1–4, 2021.

[34] H. D. Prasetyo, P. A. Hogantara, I. F. Nurahmadan, R. M. Arjuna, I. N. Isnainiyah, and R. Wirawan, "Cnn architecture on distinguishing art and photo: A comparison," in *Proceedings of the 3rd International Conference on Informatics, Multimedia, Cyber, and Information System (ICIMCIS)*, pp. 59–64, 2021.

[35] S. Poomrittigul, W. Chomkwah, T. Tanpatanan, S. Sakorntanant, and T. Treebupachatsakul, "A comparison of deep learning cnn architecture models for classifying bacteria," in *ITC-CSCC 2022 - 37th International Technical Conference on Circuits/Systems, Computers and Communications*, pp. 290–293, 2022.

[36] J. Y. Ahn and N. I. Cho, "Multi-branch neural architecture search for lightweight image super-resolution," *IEEE Access*, vol. 9, pp. 153633–153646, 2021.

[37] T. Fukagai *et al.*, "Speed-up of object detection neural network with gpu," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, (Athens, Greece), pp. 301–305, 2018.

[38] Y. Liu, "An improved faster r-cnn for object detection," in *Proceedings of the 2018 11th International Symposium on Computational Intelligence and Design (ISCID)*, pp. 119–123, 2018.

[39] D. Seo, J. Ahn, and H. Nam, "Comparison of cnn architectures using rp algorithm for burst signal detection," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, (Jeju, Korea (South)), pp. 809–812, 2020.

[40] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and Quantization for Deep Neural Network Acceleration: A Survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[41] M. K. N R, S. Joshi, P. P. Chanduri, and S. S, "Comparative analysis of handwritten digit recognition using mlp, cnn, lenet-5 model's," in *2024 International Conference on Distributed Systems, Computer Networks and Cybersecurity (ICDSCNC)*, pp. 1–5, 2024.

[42] T. Soni, "Deep learning-based identification of strawberry leaf diseases with lenet architecture," in *2024 4th International Conference on Technological Advancements in Computational Sciences (ICTACS)*, pp. 978–982, 2024.

[43] S. Tan and Z. Tan, "Improved lenet-5 model based on handwritten numeral recognition," in *2019 Chinese Control And Decision Conference (CCDC)*, pp. 6396–6399, 2019.

[44] M. S. Devi, H. K. C, J. S. V, J. K, I. J. G A, and F. M. D, "Regularized dense layered lenet towards mushroom type classification," in *2024 5th International Conference for Emerging Technology (INCET)*, pp. 1–5, 2024.

[45] C. Yu, Y.-H. Li, Y.-H. Tseng, and Y.-M. Chen, "Low-latency lenet-5 architecture for handwritten digit recognition," in *2023 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pp. 1–3, 2023.

[46] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," *CoRR*, vol. abs/1507.06228, 2015.

[47] V. Gadi, P. Seth, and E. Poongothai, "Diabetic retinopathy detection using resnet-50 and resnet-152 deep learning approach," in *2025 2nd International Conference on Research Methodologies in Knowledge Management, Artificial Intelligence and Telecommunication Engineering (RMKMATE)*, pp. 1–6, 2025.

[48] S. Y. Prasetyo, "Optimizing rice variety recognition through transfer learning with pretrained resnet models," in *2024 4th International Conference of Science and Information Technology in Smart Administration (ICSINTESA)*, pp. 231–235, 2024.

[49] D. Li, P. Liu, T. Sun, L. Li, and Y. Xue, "Real-time pvc recognition system design based on multi-parameter se-resnet," *IEEE Access*, vol. 12, pp. 70345–70356, 2024.

[50] Y. Han, Z. Nan, S. Zhou, and Z. Niu, "Dvfs-aware dnn inference on gpus: Latency modeling and performance analysis," 2025.

[51] S. Zhang, S. Zhang, Z. Qian, J. Wu, Y. Jin, and S. Lu, "Deepslicing: Collaborative and adaptive cnn inference with low latency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2175–2187, 2021.

[52] W. Niu, X. Ma, Y. Wang, and B. Ren, "26ms inference time for resnet-50: Towards real-time execution of all dnns on smartphone," 2019.

[53] M. Chacon-Chamorro, F. A. Gallego, and J. Riano-Rojas, "Reducing overfitting in resnet with adaptive lipschitz regularization," *Journal of Computational and Applied Mathematics*, vol. 471, p. 116747, 2026.

[54] D. Kusumawati, A. A. Ilham, A. Achmad, and I. Nurtanio, "Vgg-16 and vgg-19 architecture models in lie detection using image processing," in *2022 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, pp. 340–345, 2022.

[55] X. Liu, H. Zhu, W. Song, J. Wang, L. Yan, and K. Wang, "Research on improved vgg-16 model based on transfer learning for acoustic

image recognition of underwater search and rescue targets," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 17, pp. 18112–18128, 2024.

[56] S. Torne, D. K. Shetty, K. Makkithaya, P. Hegde, M. Sudhi, P. Kumar Pullela, T. Tamil Eniyan, R. Kamath, S. Salu, P. Bhat, S. Girisha, and P. S. Priya, "Vgg-16, vgg-16 with random forest, resnet50 with svm, and efficientnetb0 with xgboost-enhancing bone fracture classification in x-ray using deep learning models," *IEEE Access*, vol. 13, pp. 25568–25577, 2025.

[57] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 1943–1955, 2016.

[58] H. Qassim, A. Verma, and D. Feinzimer, "Compressed residual-vgg16 cnn model for big data places image recognition," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 169–175, 2018.

[59] J. Yang, Z. Li, Z. Gu, and W. Li, "Research on floating object classification algorithm based on convolutional neural network," *Scientific Reports*, vol. 14, p. 32086, Dec 2024.

[60] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[61] J. Padhi, L. Korada, A. Dash, P. K. Sethy, S. K. Behera, and A. Nanthaamornphong, "Paddy leaf disease classification using efficientnet b4 with compound scaling and swish activation: A deep learning approach," *IEEE Access*, vol. 12, pp. 126426–126437, 2024.

[62] G. Kaur, N. Sharma, R. Chauhan, A. Garg, and R. Gupta, "Early detection of pneumothorax using efficientnet b0 transfer learning model," in *2023 2nd International Conference on Futuristic Technologies (INCOFT)*, pp. 1–4, 2023.

[63] Z. Liu, J. John, and E. Agu, "Diabetic foot ulcer ischemia and infection classification using efficientnet deep learning models," *IEEE Open Journal of Engineering in Medicine and Biology*, vol. 3, pp. 189–201, 2022.

[64] A. Batool and Y.-C. Byun, "Lightweight efficientnetb3 model based on depthwise separable convolutions for enhancing classification of leukemia white blood cell images," *IEEE Access*, vol. 11, pp. 37203–37215, 2023.

[65] R. Preetha, M. J. P. Priyadarsini, and J. S. Nisha, "Automated brain tumor detection from magnetic resonance images using fine-tuned efficientnet-b4 convolutional neural network," *IEEE Access*, vol. 12, pp. 112181–112195, 2024.

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

[67] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," 2017.

[68] 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," 2016.

[69] Z. Huang, A. Qie, C. Zhang, J. Yang, and X. Wang, "Exploration for efficient depthwise separable convolution networks deployment on

fpga," in *2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS)*, pp. 213–217, 2024.

[70] R. Yang, Z. Chen, L. Hu, X. Cui, and Y. Guo, "An fpga-based approach for compressing and accelerating depthwise separable convolution," *IEEE Signal Processing Letters*, vol. 31, pp. 2590–2594, 2024.

[71] S. Pan, X. Cheng, and W. Fan, "Classification of petrographic thin section images with depthwise separable convolution and dilated convolution," *IEEE Access*, vol. 13, pp. 28789–28798, 2025.

[72] L. F. Brillet *et al.*, "Speeding-up cnn inference through dimensionality reduction," in *2019 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, (Montreal, QC, Canada), pp. 1–6, 2019.

[73] C. Wang, T. Endo, T. Hirofuchi, and T. Ikegami, "Speed-up single shot detector on gpu with cuda," in *2022 23rd ACIS International Summer Virtual Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Summer)*, (Kyoto City, Japan), pp. 36–41, 2022.

[74] G. V. Stoica, R. Dogaru, and E. C. Stoica, "Speeding-up image processing in reaction-diffusion cellular neural networks using cuda-enabled gpu platforms," in *2014 6th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, (Bucharest, Romania), pp. 39–42, 2014.

[75] J. Pasquet, M. Chaumont, G. Subsol, and M. Derras, "Speeding-up a convolutional neural network by connecting an svm network," in *2016 IEEE International Conference on Image Processing (ICIP)*, (Phoenix, AZ, USA), pp. 2286–2290, 2016.

[76] Y. Yu, J. Jiang, and X. Chi, "Using supercomputer to speed up neural network training," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, (Wuhan, China), pp. 942–947, 2016.

[77] W. Cao, J. Yuan, Z. He, Z. Zhang, and Z. He, "Fast deep neural networks with knowledge guided training and predicted regions of interest for real-time video object detection," *IEEE Access*, vol. 6, pp. 8990–8999, 2018.

[78] J. Pang, C. Li, J. Shi, Z. Xu, and H. Feng, "R2-cnn: Fast tiny object detection in large-scale remote sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 8, pp. 5512–5524, 2019.

[79] L. L. Zhang *et al.*, "Nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *MobiSys 2021 - Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 81–93, Association for Computing Machinery, Inc, 2021.

[80] D. Cireşan and U. Meier, "Multi-Column Deep Neural Networks for Offline Handwritten Chinese Character Classification," in *2015 International Joint Conference on Neural Networks (IJCNN)*, (Killarney, Ireland), pp. 1–6, IEEE, 2015.

[81] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "High-Performance Neural Networks for Visual Object Classification," 2011.

[82] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analy-

sis," in *Seventh International Conference on Document Analysis and Recognition (ICDAR)*, (Edinburgh, UK), pp. 958–963, IEEE, 2003.

[83] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, Sept. 2014.

[84] R. Lin, "Analysis on the Selection of the Appropriate Batch Size in CNN Neural Network," in *International Conference on Machine Learning and Knowledge Engineering (MLKE)*, (Guilin, China), pp. 106–109, 2022.

[85] R. Saeed, R. Ghnemat, G. Benbrahim, and A. Elhassan, "Learning with Dynamic Architectures for Artificial Neural Networks - Adaptive Batch Size Approach," in *2nd International Conference on New Trends in Computing Sciences (ICTCS)*, (Amman, Jordan), pp. 1–4, 2019.

[86] J. Ren, D. Xu, S. Yang, J. Zhao, Z. Li, C. Navasca, C. Wang, H. Xu, and D. Li, "Enabling large dynamic neural network training with learning-based memory management," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 788–802, 2024.

[87] L. Chen, H. Wang, J. Zhao, P. Koutris, and D. Papailiopoulos, "The effect of network width on the performance of large-batch training," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, (Red Hook, NY, USA), p. 9322–9332, Curran Associates Inc., 2018.

[88] W. Chen, X. Dong, F. Zhang, B. Li, Y. Wang, and Q. Wang, "Atp: Achieving throughput peak for dnn training via smart gpu memory management," *ACM Trans. Archit. Code Optim.*, vol. 22, Mar. 2025.

[89] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," 2017.

[90] W. Chen, X. Dong, X. Chen, S. Liu, Q. Xia, and Q. Wang, "pommdnn: Performance optimal gpu memory management for deep neural network training," *Future Generation Computer Systems*, vol. 152, pp. 160–169, 2024.

[91] S. Lee, C. He, and S. Avestimehr, "Achieving small-batch accuracy with large-batch scalability via hessian-aware learning rate adjustment," *Neural Networks*, vol. 158, pp. 1–14, 2023.

[92] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," 2018.

[93] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training bert in 76 minutes," 2020.

[94] K. Do, M.-D. Nguyen, N. T. Hoa, L. Tran-Thanh, N. H. Tran, and Q.-V. Pham, "Revisiting lars for large batch training generalization of neural networks," *IEEE Transactions on Artificial Intelligence*, vol. 6, no. 5, pp. 1321–1333, 2025.

[95] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5987–5995, 2017.

[96] S. H. Mozafari, J. J. Clark, W. J. Gross, and B. H. Meyer, "Efficient 1d grouped convolution for pytorch a case study: Fast on-device

fine-tuning for squeezebert," in *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 69–75, 2023.

[97] T. Chen, B. Duan, Q. Sun, M. Zhang, G. Li, H. Geng, Q. Zhang, and B. Yu, "An efficient sharing grouped convolution via bayesian learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 7367–7379, 2022.

[98] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[99] Y. An, Y. Feng, N. Yuan, Z. Ji, and I. Ganchev, "Dirbw-net: An improved inverted residual network model for underwater image enhancement," *IEEE Access*, vol. 12, pp. 75474–75482, 2024.

[100] B. Liu, Q. Jiang, P. Wang, S. Yao, W. Zhou, and X. Jin, "Irmsd-yolo: Multiscale dilated network with inverted residuals for infrared small target detection," *IEEE Sensors Journal*, vol. 25, no. 9, pp. 16006–16019, 2025.

[101] R. Wang, H. Zhang, M. Xu, F. Zhou, and Q. Wu, "A novel lightweight automatic modulation classification scheme based on inverted residuals," in *2023 International Conference on Ubiquitous Communication (Ucom)*, pp. 259–263, 2023.

[102] J. Wang, H. Zou, M. Zhou, and R. Su, "An efficient deep neural network for surface defect detection in industrial edge sensing," *IEEE Transactions on Industrial Informatics*, vol. 21, no. 3, pp. 2560–2569, 2025.

[103] S.-W. Lee, J.-M. Sung, and J.-W. Kwon, "Optimizing rgb convolution on cortex-m7 mcu: Approaches for performance improvement," *IEEE Access*, vol. 12, pp. 140594–140604, 2024.

[104] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," 2020.

[105] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2017.

[106] V. Radu, K. Kaszyk, Y. Wen, J. Turner, J. Cano, E. J. Crowley, B. Franke, A. Storkey, and M. O'Boyle, "Performance aware convolutional neural network channel pruning for embedded gpus," 2020.

[107] H. Kong, D. Liu, X. Luo, S. Huai, R. Subramaniam, C. Makaya, Q. Lin, and W. Liu, "Towards efficient convolutional neural network for embedded hardware via multi-dimensional pruning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2023.

[108] X. Cai, Y. Wang, and L. Zhang, "Optimus: An operator fusion framework for deep neural networks," *ACM Trans. Embed. Comput. Syst.*, vol. 22, Oct. 2022.

[109] M. Yoon and J. Choi, "Architecture-aware optimization of layer fusion for latency-optimal cnn inference," in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 1–4, 2023.

[110] J. I. Mestre, S. Barrachina, D. Quezada, and M. F. Dolz, "Deep learning inference optimisation for iot: Conv2d-relu-bn layer fusion and quantisation," *The Journal of Supercomputing*, vol. 81, p. 621, Mar 2025.

[111] S. Goel, R. Kedia, R. Sen, and M. Balakrishnan, "EXPRESS: CNN EXecution Time PREdiction for DPU DeSign Space Exploration," in *FPT 2022 - 21st International Conference on Field-Programmable Technology*, IEEE, 2022.

[112] A. Hazarika, S. Poddar, M. M. Nasralla, and H. Rahaman, "Area and Energy Efficient Shift and Accumulator Unit for Object Detection in IoT Applications," *Alexandria Engineering Journal*, vol. 61, pp. 795–809, Jan. 2022.

[113] E. Yago, P. Castello, S. Petit, M. E. Gomez, and J. Sahuquillo, "Impact of the array shape and memory bandwidth on the execution time of cnn systolic arrays," in *Proceedings - Euromicro Conference on Digital System Design, DSD 2020*, pp. 510–517, IEEE, 2020.

[114] T. Garbay *et al.*, "CNN Inference Costs Estimation on Microcontrollers: The EST Primitive-based Model," in *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, IEEE, 2021.

[115] K. Desappan, M. Mody, M. Mathew, P. Swami, and P. Eppa, "CNN Inference: Dynamic and Predictive Quantization," in *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, (Berlin, Germany), pp. 1–4, 2018.

[116] Z. Zhang *et al.*, "RTCoInfer: Real-Time Collaborative CNN Inference for Stream Analytics on Ubiquitous Images," *IEEE Journal on Selected Areas in Communications*, vol. 41, pp. 1212–1226, Apr. 2023.

[117] M. D. Dere, J. Ji-Hun, and B. Lee, "Towards On-device Deep Neural Network Inference and Model Update for Real-time Gesture Classification," in *2022 22nd International Conference on Control, Automation and Systems (ICCAS)*, (Jeju, Korea), pp. 337–341, 2022.

[118] N. Li, A. Iosifidis, and Q. Zhang, "Receptive Field-based Segmentation for Distributed CNN Inference Acceleration in Collaborative Edge Computing," in *ICC 2022 - IEEE International Conference on Communications*, (Seoul, Korea), pp. 4281–4286, 2022.

[119] H. Kim, T. Kang, and K. Choi, "A Study on the Supplementary Method between RNN and CNN Model to Improve the Accuracy of Fine Dust Concentration Inference," in *13th International Conference on Information and Communication Technology Convergence (ICTC)*, (Jeju Island, Korea), pp. 389–394, 2022.

[120] W. Lan, J. Dang, Y. Wang, and S. Wang, "Pedestrian Detection Based on YOLO Network Model," in *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, (Changchun, China), pp. 1547–1551, IEEE, 2018.

[121] J. Hu, X. Gao, H. Wu, and S. Gao, "Detection of Workers Without the Helmets in Videos Based on YOLO V3," in *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, (Suzhou, China), pp. 1–4, IEEE, 2019.

[122] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1137–1149, June 2017.

[123] C. Guo *et al.*, "Accelerating Sparse DNN Models without Hardware-Support via Tile-Wise Sparsity," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, (Atlanta, GA, USA), pp. 1–15, IEEE, 2020.

[124] S. Yang *et al.*, "APQ: Automated DNN Pruning and Quantization for ReRAM-Based Accelerators," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, pp. 2498–2511, Sept. 2023.

[125] S. Liu and W. Deng, "Very Deep Convolutional Neural Network-Based Image Classification Using Small Training Sample Size," in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, (Kuala Lumpur, Malaysia), pp. 730–734, IEEE, 2015.

[126] B. McDanel, H. Dinh, and J. Magallanes, "Accelerating DNN Training with Structured Data Gradient Pruning," in *2022 26th International Conference on Pattern Recognition (ICPR)*, (Montreal, QC, Canada), pp. 2293–2299, IEEE, 2022.

[127] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[128] S. Green, C. M. Vineyard, and K. Koç, "Mathematical Optimizations for Deep Learning," in *Cyber-Physical Systems Security*, pp. 69–92, Springer, 2018.

[129] W. Shi, Y. Gong, X. Tao, D. Cheng, and N. Zheng, "Fine-Grained Image Classification Using Modified DCNNs Trained by Cascaded Softmax and Generalized Large-Margin Losses," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, pp. 683–694, Mar. 2019.

[130] "Conv2d - PyTorch 2.2 Documentation." https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d. Accessed: Mar. 4, 2024.

[131] S. Goel, R. Kedia, M. Balakrishnan, and R. Sen, "INFER: INterFerence-aware Estimation of Runtime for Concurrent CNN Execution on DPUs," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, (Maui, HI, USA), pp. 66–71, IEEE, 2020.

[132] J. Chang, Y. Choi, T. Lee, and J. Cho, "Reducing MAC Operation in Convolutional Neural Network with Sign Prediction," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, (Jeju, Korea (South)), pp. 177–182, IEEE, 2018.

[133] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," in *International Conference on Learning Representations*, 2016.

[134] K. Fukushima, "Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements," *IEEE Transactions on Systems Science and Cybernetics*, vol. 5, pp. 322–333, Oct. 1969.

[135] L. Lu, "Dying ReLU and Initialization: Theory and Numerical Examples," *Communications in Computational Physics*, vol. 28, pp. 1671–1706, June 2020.

[136] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *International Conference on Machine Learning*, 2015.

[137] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How Does Batch Normalization Help Optimization?," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, (Montréal, Canada), pp. 2488–2498, 2018.

[138] P. Baldi and P. Sadowski, "Understanding Dropout," in *Advances in Neural Information Processing Systems*, vol. 26, 2013.

[139] S. Wager, S. Wang, and P. Liang, "Dropout Training as Adaptive Regularization," *Advances in Neural Information Processing Systems*, 2013.

[140] K. Liu, G. Kang, N. Zhang, and B. Hou, "Breast Cancer Classification Based on Fully Connected Layer First Convolutional Neural Networks," *IEEE Access*, vol. 6, pp. 23722–23732, 2018.

[141] J. S. Bridle, "Training Stochastic Model Recognition Algorithms as Networks Can Lead to Maximum Mutual Information Estimation of Parameters," in *Advances in Neural Information Processing Systems*, vol. 2, pp. 211–217, Jan. 1989.

[142] S. Montaha, S. Azam, A. K. M. R. H. Rafid, M. Z. Hasan, A. Karim, and A. Islam, "TimeDistributed-CNN-LSTM: A Hybrid Approach Combining CNN and LSTM to Classify Brain Tumor on 3D MRI Scans Performing Ablation Study," *IEEE Access*, vol. 10, pp. 60039–60059, 2022.

[143] H. Xiao, H. Xu, X. Chen, Y. Wang, and Y. Han, "Fast and High-Accuracy Approximate MAC Unit Design for CNN Computing," *IEEE Embedded Systems Letters*, vol. 14, pp. 155–158, Sept. 2022.

[144] S. Moini, B. Alizadeh, M. Emad, and R. Ebrahimpour, "A Resource-Limited Hardware Accelerator for Convolutional Neural Networks in Embedded Vision Applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, pp. 1217–1221, Oct. 2017.

[145] H. O. Ahmed, M. Ghoneima, and M. Dessouky, "Concurrent MAC Unit Design Using VHDL for Deep Learning Networks on FPGA,"

in *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, (Penang, Malaysia), pp. 31–36, 2018.

[146] J. Ma and Z. Salcic, "Deep Learning with Accelerated Execution: A Real-Time License Plate Localisation System," in *2021 IEEE 19th International Conference on Industrial Informatics (INDIN)*, (Palma de Mallorca, Spain), pp. 1–8, 2021.

[147] P. Song, Y. He, L. Zhang, G. Zhang, Y. Chen, W. Wang, B. Tian, and Y. Xie, "An Improved Cascade R-CNN and RGB-D Camera-Based Method for Dynamic Cotton Top Bud Recognition and Localization in the Field," *Computers and Electronics in Agriculture*, vol. 202, p. 107442, Nov. 2022.

[148] Y. Wu, W. Liu, and S. Wan, "Multiple Attention Encoded Cascade R-CNN for Scene Text Detection," *Journal of Visual Communication and Image Representation*, vol. 80, p. 103261, Oct. 2021.

[149] K. Wang, Y. Dong, H. Bai, Y. Zhao, and K. Hu, "Use Fast R-CNN and Cascade Structure for Face Detection," in *2016 Visual Communications and Image Processing (VCIP)*, (Chengdu, China), pp. 1–4, 2016.

[150] M. Thilagavathi and S. Abirami, "Cascade-Forward Neural Network in Identification of Plant Species of Desert Based on Wildflowers," in *2018 IEEE International Conference on System, Computation, Automation and Networking (ICSCA)*, (Pondicherry, India), pp. 1–7, 2018.

[151] R. R. O. Al-Nima, F. H. Abdulraheem, and M. Y. Al-Ridha, "Using Hand-Dorsal Images to Reproduce Face Images by Applying Back Propagation and Cascade-Forward Neural Networks," in *2019 2nd*

International Conference on Electrical, Communication, Computer, Power and Control Engineering (ICECCPCE), (Mosul, Iraq), pp. 97–102, 2019.

[152] Y. Li and Z. Wang, "Research on Textile Defect Detection Based on Improved Cascade R-CNN," in *2021 International Conference on Artificial Intelligence and Electromechanical Automation (AIEA)*, (Guangzhou, China), pp. 43–46, 2021.

[153] Z. Cai and N. Vasconcelos, "Cascade R-CNN: High Quality Object Detection and Instance Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019.

[154] H. Dong, Y. Li, and R. Liu, "A Detection Algorithm Based on Improved Cascade R-CNN for UAV Aerial Images," in *2023 IEEE 3rd International Conference on Electronic Technology, Communication and Information (ICETCI)*, (Changchun, China), pp. 700–704, 2023.

[155] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for mobilenetv3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1314–1324, 2019.

[156] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[157] Y. Chen, J. Lou, M. Wabnitz, J. Loh, and T. Gemmeke, "Edea: Efficient dual-engine accelerator for depthwise separable convolution with direct data transfer," in *2024 IEEE 37th International System-on-Chip Conference (SOCC)*, pp. 1–6, 2024.

[158] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021.

[159] E. J. Husom, A. Goknil, M. Astekin, L. K. Shar, A. Kåsen, S. Sen, B. A. Mithassel, and A. Soylu, "Sustainable llm inference for edge ai: Evaluating quantized llms for energy efficiency, output accuracy, and inference latency," 2025.

[160] D. Rossi, G. Borghi, and R. Vezzani, " TakuNet: An Energy-Efficient CNN for Real-Time Inference on Embedded UAV Systems in Emergency Response Scenarios ," in *2025 IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*, (Los Alamitos, CA, USA), pp. 339–348, IEEE Computer Society, Mar. 2025.

[161] S. Shakibhamedan, N. Amirafshar, A. S. Baroughi, H. S. Shahhoseini, and N. TaheriNejad, "Ace-cnn: Approximate carry disregard multipliers for energy-efficient cnn-based image classification," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 5, pp. 2280–2293, 2024.

[162] R.-Y. Ju, T.-Y. Lin, J.-H. Jian, and J.-S. Chiang, "Efficient convolutional neural networks on raspberry pi for image classification," 2022.

[163] X. Sui, Q. Lv, L. Zhi, B. Zhu, Y. Yang, Y. Zhang, and Z. Tan, "A hardware-friendly high-precision cnn pruning method and its fpga implementation," *Sensors*, vol. 23, no. 2, 2023.

[164] S. Naveen and M. R. Kounte, "Optimized convolutional neural network at the iot edge for image detection using pruning and quanti-

zation," *Multimedia Tools and Applications*, vol. 84, pp. 5435–5455, Mar 2025.

[165] B. Liu, Z. Yin, Z. Duan, J. Xiao, Y. Tan, J. Wang, Z. Shen, Z. Wu, and C. Wang, "A fully pipelined, low-overhead, and energy-efficient cnn-based feature extraction accelerator for mobile visual slam," *Microelectronics Journal*, vol. 165, p. 106860, 2025.

[166] A. Jayasimhan and P. P., "Resprune: An energy-efficient restorative filter pruning method using stochastic optimization for accelerating cnn," *Pattern Recognition*, vol. 155, p. 110671, 2024.

[167] A. Ghanbari and M. Modarressi, "Energy-efficient acceleration of convolutional neural networks using computation reuse," *Journal of Systems Architecture*, vol. 126, p. 102490, 2022.

[168] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[169] Z. Cai and N. Vasconcelos, "Cascade r-cnn: Delving into high quality object detection," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6154–6162, 2018.

[170] M. Ghasemi, D. Rakhmatov, C.-J. Wu, and S. Vrudhula, "Edgewise: Energy-efficient cnn computation on edge devices under stochastic communication delays," *ACM Trans. Embed. Comput. Syst.*, vol. 21, Oct. 2022.

[171] A. Kouris, S. I. Venieris, and C.-S. Bouganis, "A throughput-latency co-optimised cascade of convolutional neural network classifiers," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1656–1661, 2020.

[172] S. Minakova and T. Stefanov, "Memory-throughput trade-off for cnn-based applications at the edge," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, Dec. 2022.

[173] T. Theodoropoulos, J. Violos, A. Makris, and K. Tserpes, "A new approach for evaluating the performance of distributed latency-sensitive services," in *2024 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 365–370, 2024.

[174] M. Kinnas, J. Violos, I. Kompatsiaris, and S. Papadopoulos, "Reducing inference energy consumption using dual complementary cnns," *Future Generation Computer Systems*, vol. 165, p. 107606, 2025.

[175] J. Hanhirova, T. Kämäräinen, S. Seppälä, M. Siekkinen, V. Hirvisalo, and A. Ylä-Jääski, "Latency and throughput characterization of convolutional neural networks for mobile computer vision," in *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, (New York, NY, USA), p. 204–215, Association for Computing Machinery, 2018.

[176] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, "Facial Landmark Detection by Deep Multi-task Learning," in *Computer Vision – ECCV 2014*, vol. 8694 of *Lecture Notes in Computer Science*, pp. 94–108, Springer, Cham, 2014.

[177] Z. Wang, J. Zhang, H. Zhang, D. Guo, Y. Zhang, and J. Zhai, "Accurate Scoliosis Vertebral Landmark Localization on X-ray Images via Shape-Constrained Multi-Stage Cascaded CNNs," *Fundamental Research*, Nov. 2022.

[178] H. R. Roth, L. Lu, A. Farag, A. Sohn, E. Turkbey, H. Shin, and R. M. Summers, "DeepOrgan: Multi-level Deep Convolutional Networks for

Automated Pancreas Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, vol. 9351 of *Lecture Notes in Computer Science*, pp. 556–564, Springer, Cham, 2015.

[179] Y. Chen, Z. Wang, Y. Peng, Z. Zhang, G. Yu, and J. Sun, "Cascaded Pyramid Network for Multi-person Pose Estimation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Salt Lake City, UT, USA), pp. 7103–7112, 2018.

[180] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A Convolutional Neural Network Cascade for Face Detection," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Boston, MA, USA), pp. 5325–5334, 2015.

[181] A. Kouris, S. I. Venieris, and C.-S. Bouganis, "Cascade cnn: Pushing the performance limits of quantisation in convolutional neural networks," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 155–1557, 2018.

[182] A. Kouris, S. I. Venieris, and C.-S. Bouganis, "A throughput-latency co-optimised cascade of convolutional neural network classifiers," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1656–1661, 2020.

[183] Z. Cai and N. Vasconcelos, "Cascade r-cnn: Delving into high quality object detection," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6154–6162, 2018.

[184] Z. Cai and N. Vasconcelos, "Cascade r-cnn: High quality object detection and instance segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, pp. 1483–1498, 2021.

[185] C.-Y. Sun, X.-J. Hong, S. Shi, Z.-Y. Shen, H.-D. Zhang, and L.-X. Zhou, "Cascade faster r-cnn detection for vulnerable plaques in oct images," *IEEE Access*, vol. 9, pp. 24697–24704, 2021.

[186] N. James, L.-Y. Ong, and M.-C. Leow, "Exploring distributed deep learning inference using raspberry pi spark cluster," *Future Internet*, vol. 14, no. 8, 2022.

[187] Z. Cai and N. Vasconcelos, "Cascade r-cnn: Delving into high quality object detection," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6154–6162, 2018.

[188] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[189] M. Wang and W. Deng, "Deep face recognition: A survey," *Neurocomputing*, vol. 429, pp. 215–244, 2021.

[190] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, pp. 115–118, Feb 2017.

[191] A. M. Salaheldin, M. Abdel Wahed, M. Talaat, and N. Saleh, "An evaluation of ai-based methods for papilledema detection in retinal fundus images," *Biomedical Signal Processing and Control*, vol. 92, p. 106120, 2024.

[192] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, pp. 611–629, Aug 2018.

[193] A. Ghahremani, E. Bondarev, and P. H. De With, "Cascaded cnn method for far object detection in outdoor surveillance," in *2018*

14th International Conference on Signal-Image Technology Internet-Based Systems (SITIS), pp. 40–47, 2018.

[194] J. Amin, M. A. Anjum, K. Ibrar, M. Sharif, S. Kadry, and R. G. Crespo, "Detection of anomaly in surveillance videos using quantum convolutional neural networks," *Image and Vision Computing*, vol. 135, p. 104710, 2023.

[195] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed Pooling for Convolutional Neural Networks," in *Rough Sets and Knowledge Technology (RSKT 2014)* (D. Miao, W. Pedrycz, D. Ślzak, G. Peters, Q. Hu, and R. Wang, eds.), vol. 8818 of *Lecture Notes in Computer Science*, pp. 364–375, Springer, Cham, 2014.

[196] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Honolulu, HI, USA), pp. 2261–2269, 2017.

[197] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, "Dual Path Networks," in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 4470–4478, 2017.

Appendices

CHAPTER A

Appendices

A.1 Performance Evaluation of a Proposed Two Stage Cascaded Convolutional Neural Network Architecture

Performance Evaluation of a Proposed Two Stage Cascaded Convolutional Neural Network Architecture

Abstract

Convolutional neural network models have been developed over the past few decades demonstrating remarkable performance in image classification tasks for various applications in industries. The potential to produce higher accuracy has resulted in deep convolutional neural networks widely used in research where the presence of complex computations is the secret of success. Larger complex computations cause an increased inference time for a single classification, which eventually leads to overall latency of vision-based systems. This research work focuses on minimizing the inference time of a convolutional neural network while achieving better classification accuracy by cascading modified convolutional neural network models. Learnable layers of a CNN, such as convolutional and fully connected layers, are involved in heavy complex computations that account for most of the inference time. Modified architecture with optimized computations while maintaining generalization capability leads to minimizing latency. The cascaded neural network model trained on a lesser number of output classes offered better classification accuracies and reduced inference time compared to standard convolutional neural network models such as Alex net. The performance of the cascaded neural network model was tested for classification accuracy and execution time for a wide range of test data presented, that proposed cascaded network inference speed was enhanced by 6% while maintaining the classification accuracy as the AlexNet for tests conducted on 200 object classes.

Keywords: CNNs, Cascaded CNN, Complex computations, Inference time, Classification accuracy

1 Introduction

During the Industrial Revolution, the development of sophisticated machines became common practice to carry out industrial activities, ensuring productivity, efficiency, and safety while replacing manual labor. As these systems needed to take timely actions with minimal or no supervision, it was essential for the system to make critical decisions independently. The detection and identification of different objects in a dynamic environment [1] is an easy task for humans, while the most complicated is

for a machine [2] [3]. This challenge inspired researchers to develop classical object classifiers. Due to the complex computations involved and straightforward results, linear classifiers were limited for practical applications. For certain applications such as mobile robots [1], service robots, medical robots, etc., with limited power and resource availability, the application of classical classifiers was a great challenge. The perceptron [4] [5] being the simplest artificial neural network developed by Rosenblatt in 1957 inspired by biological neurons, formulated the very first linear classifier for supervised tasks. With the rigorous research involved over the years with the development of technologies and devices, a more flexible classification model based on the convolutional neural network, LeNet-5 [5] was developed by LeCun et al. in the late 1990s.

LeNet-5 is a simple network architecture that consists of five convolutional layers. Originally designed for the detection of handwritten numbers, the network had trained on the MNIST data set [6] inspired AlexNet by Krizhevsky et al. in 2012. The data set for LeNet-5 was an unnormalized data example of handwritten data.AlexNet Being a turning point in machine learning,[7]had implemented more dense layers in the network architecture compared to the original LeNet-5 with just five layers. The winner of the ILSVRC-2012 challenge, AlexNet, has been trained on ImageNet data set with a wide variety of data over 1000 object classes, motivated researchers to use more dense networks for better generalization, which could be seen in models such as ResNet [5] where there is a lack of connectivity. Alexnet's impressive classification performance on ImageNet dataset was well explained in [8] proposing the de convolution method citing the Alexnet compatibility on small training sets. Although there was a trend to classify more objects with higher accuracy using more complex computations involved, another pragmatic approach was to speed up the process with reduced filter size, which has been adapted in [9][10].However, the main intention of reducing filter size and strides was achieved in [8], and was further generalized in GoogLeNet. Another novel approach to achieve better classification with faster processing was the use of single-shot detectors, where Redmon et al. adapted reduced filter sizes in YOLO [9]. Filter size strides and Darknet-19 output sizes, which was based on YOLO are presented in Table 1. The impressive results of [9] and [5] motivated researchers to apply the models in [10] for the automated detection of melanoma. A region proposal network model [11] was adapted in Faster-RCNN which also proposes faster execution compared to previous network models. The successful detection method of pedestrians [12] has been proposed by Wenbo Lan et al., by adapting the above methods. A deep learning method has been adapted, as the name suggests, in which more deep or dense layers are present in their network architectures. It is important to know that the dense and sparse connectivity present within a CNN model has a significant impact on the classification speed and accuracy. Although the detection accuracy has been improved, interestingly, the processing speed has been enhanced to 25 frames per second with the modified YOLO architecture, producing a new model YOLO-R. Similarly, a real-time people counting application [13]proposed by Peiming et al., which involves counting people in real-time dynamic environments. The need of fast execution of the network identified by the research community proposes novel approaches such as detection of workers without safety helmet in construction site real-time video achieving a fast processing as fast as 35 frames per second.

Table 1: DarkNet specifications

Type	Filters	Size/stride	Output
Convolutional	32	3 x 3	224 x 224
Maxpool		2 x 2/2	112 x 112
Convolutional	64	3 x 3	112 x 112
Maxpool		2 x 2/2	56 x 56
Convolutional	128	3 x 3	56 x 56
Convolutional	64	1 x 1	56 x 56
Convolutional	128	3 x 3	56 x 56
Maxpool		2 x 2/2	28 x 28
Convolutional	256	3 x 3	28 x 28
Convolutional	128	1 x 1	28 x 28
Convolutional	256	3 x 3	28 x 28
Maxpool		2 x 2/2	14 x 14
Convolutional	512	3 x 3	14 x 14
Convolutional	256	1 x 1	14 x 14
Convolutional	512	3 x 3	14 x 14
Convolutional	256	1 x 1	14 x 14
Convolutional	512	3 x 3	14 x 14
Maxpool		2 x 2/2	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional	512	1 x 1	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional	512	1 x 1	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional	1000	1 x 1	7 x 7
Avgpool	1	Global	1000
Softmax			

2 Related Work

2.1 Factors affecting the CNN execution speed

Execution speed of a CNN model can be improved by various different methods that have been carried out in various researches. A general survey on CNN optimization in [14] describes common optimizations. As discussed in the Introduction, the most important operation of any convolutional neural network occurs in the sparse connectivity populated by the convolutional units and dense layers where the neurons converge. In sparse connectivity, neurons of CNN layer are connected to the subset of neurons from the previous layer. Sparsity in CNNs is achieved through the use of convolutional operations, where kernels slide over the input to extract features. Reduction in the number of parameters and local receptive fields of sparse connectivity leads to computational efficiency. Adaptation of the Inception module with small kernel sizes executed in parallel has further enhanced the performance of the CNN model. It is important to know the limitations associated with the standard convolutional neural networks. The most significant challenge associated with CNN's is computational complexity. CNNs are computationally intensive due to the large number of learnable parameters. Deeper architectures with more convolutional layers are involved in large number of mathematical computations, require dedicated hardware such as

graphical processing units (GPU), tensor processing units (TPU) or neural processing units(NPU) to perform extensive computations. Training of deeper CNN models requires a large amount of electrical power, which results in more energy consumption. Another challenge associated with CNN is overfitting. Convolutional neural networks are more prone to overfitting for training on small datasets. Mathematically complex computations are being taken at these two stages, which contributes a significant amount of time consumption by the network model. In general,factors that affect the processing speed of a CNN can be classified into two broad categories such that

1. Hardware resource oriented factors.
2. Model architecture related factors.

The factors related to the model architecture can be explained as follows. Model architecture depth: For a given network, if there are more sparse connections [15] and dense connections, this model is known as a deep network. Deep convolutional networks introduced by Fukushima [4], where the "neocognitron" consists of a cascaded interconnection of layers in the network model. The architecture has been improved in [7] and further simplified in [16] and [17]. Due to the presence of more convolutional layers and dense layers in a network, more complex computations are to be executed, which generally requires more time for the computation and thus longer inference time. Reducing the depth of the network thus leads to fast inference while maintaining fair accuracy. Convolutional layer type: Within the convolutional layer, a large number of computations are performed. Size of the Kernel[18] and strides determine the number of complex computations involved in the network.The kernel size is one of the factors that contributes to the number of computations. Thus,inference depends on the number of convolutional layers present in the network. The importance of kernel size extensively discussed in GoogLeNet [19] architecture, where the inception module presented in Fig. 1 is a remarkable contribution. Pooling layers in network architecture: The pooling layers are used after the convolutional layers, essentially after the activations to reduce spatial dimensions before propagating to the next layer thus reducing the computations. Pooling layers must be carefully selected as it leads to loss of information if excessively applied. Model weight pruning: reducing unimportant weights results in reduction of computations involved during the network execution and thereby reducing the inference. However, it is also possible to remove less important filters which further reduces model size. As it may lead to the accuracy of the network it has to be carefully conducted. Hardware resource-oriented factors which affect the speed can be further explained as bellows. Hardware acceleration devices: Using specialized hardware tools the process of execution can be accelerated. Graphical Processing Units (GPU) [16] or Tensor Processing Units (TPU) are the latest accelerators widely used to speed up convolutional neural networks inference. Also, these devices facilitate parallel computation where the workload is distributed among the cores. Batch sizing: When processing in parallel it is required to arrange the data in batches [20]. Batch size has better advantage over inference since processing is speed up with more batch size. However, there is a disadvantage with increasing batch size since it requires large memory [21] which is expensive. Hence the is a trade-off between batch size and computation efficiency. Input size and format: Since the image input is

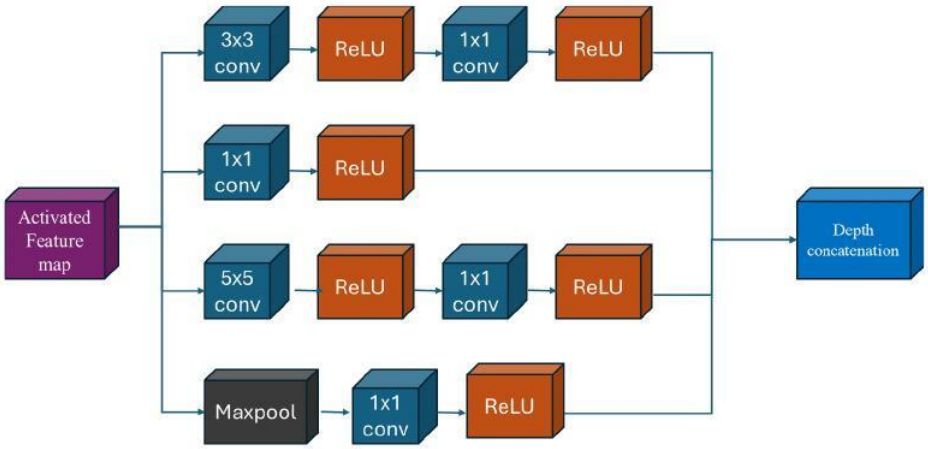


Fig. 1: Implementation of inception Module

the only source of information, alteration of the input results in classification accuracy [22] as it may lead to distortion of the original data. Reducing the input image size speeds up the execution of the network while resulting in loss of information. Caching and memorization: the intermediate results should be stored which reduces redundant computations. This leads to speeding up the execution process. However, cache memory is an expensive option as the resources should be changed. Therefore, selection of cache must be optimum in terms of performance.

2.2 Options for the reduction of the inference time

Considering the factors contribute to the longest execution time, few options can be highlighted to improve the speed of execution of a CNN model such as executing resource changes, minimizing the learnable layers, minimizing kernel sizes etc. Out of these possible methods the most suitable and optimum solution is the reduction of complex computations involved [2]. Though other options affect the speed of execution, due to the feasibility of implementation and overall performance enhancement, other methods are not the best options. Improvement of the computational resources can be achieved through switching to higher processors [12] with higher clock rates and large cache memory. However, it is not an overall feasible option as there could be systems with mobile computation platforms with low computation power such as Raspberry pi [9]. Model architectural modifications: the execution time can be reduced significantly by reducing the number of layers [2] and parameters involved during the complex computation steps. Ideally the convolutional layers and hidden layers contribute to the largest inference. However unnecessary reduction of layers may lead to the degradation of the accuracy of the model and thus the selection should be wisely made with trial and error as there's no rule to be followed. Choosing an efficient

architecture with guaranteed performance is a wise selection before any convolutional neural network related activity. Model filter pruning: reducing filter parameters such as weights results in reduction of computation burden. If unnecessary filters are present, removal of such filters reduces the model size and thus the overall computations. Kernel optimizations: using optimum kernels compatible with deep learning libraries and hardware specific libraries results in speeding up the process as the optimum computations are selected by the libraries. The hardware resources can also be changed to enhance the processing speed [12][23] of a convolutional neural network. Due to the physical component changes it is an expensive option for speed enhancement. However most relevant hardware resource related modifications are presented below. Application of hardware acceleration: rather standalone central processing unit, a graphical processing unit GPU [12][15] or a tensor processing unit TPU which offers more computation power remarkably reduces the time consumed for processing complex computations involved. The presence of such devices not only speeds up the execution of the CNN but also reduces the time taken during training of the neural network model. Batch processing: During the execution of the network model, processing the data in larger batches reduces the time consumption significantly. It is advantageous in parallel execution on hardware through techniques such as multi-threading and multi-core processing. However, batch processing requires adequate memory allocation which is another expensive option. Dedicated inference engines and hardware: certain hardware platforms are blended with sophisticated inference engines such as NVIDIA [12]. Use of such inference engines which are optimized can accelerate the computations and thereby reduce inference time. Upgrading the computation hardware is also an expensive option where overall performance can be enhanced. Parallelism: Presence of multiple computation devices or multiple CPUs [3] can offer two types of parallelism. Model parallelism, where a large network model can be split into multiple processors and process different sections of the network model concurrently. Data parallelism on the other hand, divides the input data across multiple devices or processor units and processes them simultaneously. Thereby the overall process will be enhanced and thus reduce inference time. However, it is necessary for multiple devices or processors to be available.

2.3 Scientific approaches for enhancing processing speed of CNNs

The root causes for the network model's execution time can be mitigated by model architectural and hardware changes. In this chapter various attempts made by researchers are discussed which supports the explanation in previous chapters. Even though hardware resource related modifications are efficient, due to the cost of operation architectural modification has been a major research trend. Dimensional reduction leads to fast processing as there were deduction of computations involved are described in [24] demonstrated the effect on dimensional reduction of CNN. In addition to quantization and model parameter pruning techniques to reduce computation complexity, principal component analysis has been adapted. Replacing filters in multiple layer network, allowing reduced weights and thus computations needed lesser storage resulting

overall enhancement on inference time. With respect to the architectural modifications [11][13][25][26], essentially convolutional and fully connected layer modifications, a novel approach of speeding up CNN by image pre-processing and post-processing [27] were developed by T. Fukagai et al. Speeding up of the classification process by reducing the exploration of the CNN [28] was developed by J. Pasquet et al where a support vector machine combined to feature maps of the CNN and thereby reducing unnecessary time taken by the CNN. Speed up of CNN used for object detection in real-time video application by combining region proposal algorithm with sophisticated network model, Faster R-CNN [29] demonstrated by T.M. Phuong and N.N. Diep. A novel approach for fast inference has been proposed by S. Durvasula et al for the application of event cameras. Reduction of computations by looking into consecutive input tensors where only small difference present for input camera and thus reducing large number of floating-point computations. Reducing unnecessary computations had led them to speed up inference up to 1.5 times. Another network model architectural change has been presented by J Pang et al for the fast detection of tiny objects in large scale remote sensing images [30]. It is important to accurately measure the inference latency of a deep neural network. L.Y Zhang et al presented a novel approach for determination of inference by dividing to kernel level [31]. The importance of hardware resources to reduce the inference time has been highlighted by Y. Yue et al for the work in speed up neural network training using supercomputer [23] where the data parallelism performed on multiple processing units. Their results comparatively demonstrate the effect on speed up of the performance as the processing units changed. A. Lavin and S. Gray have extensively discussed the importance of Graphical Processing Unit and optimized libraries developed novel fast algorithm [11] for the convolutional neural networks . A similar approach can be seen on fast convolutional network evaluation in [12] by N. Vasilache et al for their comparison of performance profile of convolutional neural networks. Their optimized library cuFFT has demonstrated 1.4 – 14.5 times speed up in CNNs. One of the possible methods to speed up the execution of CNN models is optimization of computations involved [32]. The performance of convolutional neural networks can be improved with optimization of computations as presented by A. Hazarika et al where the computations involved considered as multiply and accumulate (MAC) operations which repeatedly appears across the different layers. Due to the presence of high precision floating point computations on training as well as inference deep neural networks are computationally expensive [33]. A novel approach to reduce computations viewed as multiply and accumulations by reducing large models into smaller models and optimizing weights [33]. Reduction in MAC operations in convolutional layers presented by J Chang et al for speedup deep neural networks resulting in 28% increase in inference speed [34] within accelerator environment. Table 2 presents comparison of different convolutional neural network models along with their learnable parameters and MAC operations for the convolutional and fully connected layers. The drawback in convolution operation as a dot product of two vectors at the convolution layers [35][36] was highlighted and the solutions presented for this issue was broadly on two streams software optimizations and hardware optimizations[37]. In terms of hardware speedup of the CNNs inference various approaches have been suggested such as FPGA based MAC unit designs [36]

Table 2: Comparison of convolutional layer (CONV) and fully connect (FC) layer computations

Model	Parameters(million)	CONV (%)	FC(%)	Operations(millions)	CONV(%)	FC(%)
AlexNet	61.0	3.8	96.2	725	91.9	8.1
VGG-16	138.0	10.6	89.4	15484	99.2	0.8
GoogLeNet	6.9	85.1	14.9	1566	99.9	0.1
ResNet	25.6	-	-	3900	-	-

for higher degree of parallelism. Another approach to speed up inference by reducing off chip memory access [35] while achieving minimal resource utilization. Design of Approximate MAC unit had been presented by Hang Xiao et al for the purpose of CNN acceleration ensuring fast and accurate results. The design was taken into consideration of distribution probabilities of weights and activations and developed on 4:2 compressor and Dadda multiplier [38]. Impact on regularization techniques have been presented by Anbananthen et al by integrating L1 and L2 regularization on YOLOv5 [39]. Surono et al have presented accuracy enhancement method combining CNN model MobileNetv2 with Bayesian optimizer [40] Even though the FPGA based CNN speedup approaches are hardware based and are not relevant for proposed network model design, the weaknesses associated with convolutional neural network were highlighted. Various research focused on parallel implementation on deep networks to reduce the inference time as multiple processors can compute concurrently. Most of the cascaded approaches [41][42][43] are best suitable for extraction of features concurrently and thereby reduce the total time for inference. K Wang et al proposed a human face detection based on fast region based convolutional neural network (Fast R-CNN) adapting PReLU [44] for multistage detectors achieving impressive results. Zhao et al have presented the impact on activation function adapting Leaky ReLU [45] on YOLOv4 algorithm. A similar application of the cascaded feed forward network can be found in [46] for the application of plant species identification using flower images. R Raid et al have adapted model parallelism as well as data parallelism for bio metric identification purposes [47] which presents attempt on adapting cascaded architectures for enhanced performance of CNNs. Another adaptation of Fast R-CNN for cascaded operation by Yu Li et al for the defect detection on fabrics in textile industry [48] with impressive accuracy in defect detection. Cascade R-CNN [49] presented by Z Cai et al has made a significant impact on multi-object detection in real time compared to other two-stage detectors and was adapted in various research [37][48][50]. Both model parallelism and data parallelism are important concepts in CNN training on large datasets. Parallelism leads to effective utilization of computational resources and producing classifications with improved accuracy. However, it does not propose an improvement in the speed up of inference. Another cascaded architecture based on ResNet presented in [51] highlighting the accuracy enhancement compared to conventional architectures. Cascaded feature extraction in [52] demonstrates the Refinement of results as a characteristic of cascaded model. However inference speed improvement is yet a challenge to achieve.

Table 3: Learnable Parameters of AlexNet

Layer	Learnable Parameters
Convolution 1	34944
Convolution 2	307456
Convolution 3	885120
Convolution 4	663936
Convolution 5	442624
Fully Connect 1	37752832
Fully Connect 2	16781312
Fully Connect 3	4097000
Total	60961128

Table 4: Learnable Parameters and execution time of CNN models

CNN model	Learnable Parameters(in millions)	Execution time (in seconds)
VGG16	138	3.804877
AlexNet	361	1.919307
ResNet-50	25.5	2.026292
Inception-V3	23.8	2.818361
GoogleNet	6.9	0.788366

3 Proposed Method

In this section we propose the cascaded convolutional neural network architecture which offers enhanced execution speed while maintaining better accuracy for object classification. The learnable parameters associated with AlexNet was presented in Table 3. Learnable Parameters for other network models such as VGG 16, ResNet50, InceptionV3 and GoogLeNet was calculated along with their average execution times which are presented in Table 4. The practical execution time of any CNN model depends on various factors such as CPU type, hardware accelerators, memory bandwidth, etc. However, as an approximate comparison, different CNN models and their execution times were tested and presented in Table 4. From the tabulated data given above, the graph in Fig. 2 was plotted. Referring to the graph, it was clearer that the time consumed during the execution of the neural network model was reduced as the learnable parameters were reduced and thus the total number of computations involved during the execution of CNN model.

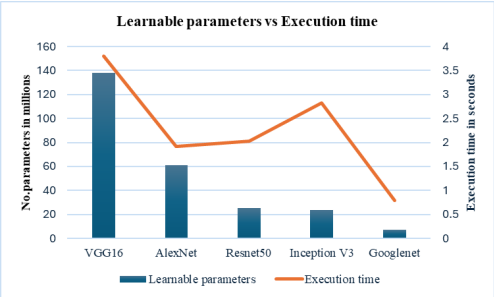


Fig. 2: Comparison of learnable parameters with execution time

To address the limitations associated with CNNs, the proposed cascaded neural network architectures are presented. The cascaded neural network model shown in Fig. 3 consists of two stages, main network and sub network. The main network consists of five consecutive convolutional layers followed by an inception module executed concurrently. The depth concatenation layer merges the feature maps to construct the input to the fully connected layer. The sub network consists of five consecutive convolutional layers, which serves as a further refinement of the main network's output. Each individual CNN model in the sub network will be selected based on the classification confidence value of the main network. The design concept was intended to be used in an airport environment for a passenger guidance service robot. Since the intended application environment has a very limited number of object classes and they can be broadly categorized into few major classes where The number of sub networks can further be increased upon requirement. The proposed design includes five sub networks, where each sub network will be activated based on the output of the main network.The network activation controller is a probabilistic mathematical function which activates the sub network based on the inference result of the main network. The mathematical functions were written in MATLAB while transfer learning was performed, and the cascaded network was executed using the deep learning toolbox. The final output interpreter is a simple mathematical function that interprets inferences from both the main network and sub network. For a given input image, the main network produces the primary classification that drives the cascaded sub network for the refinement of the classification. The main network and the sub network are computationally optimized, offering better generalization and fast inference for a given input.

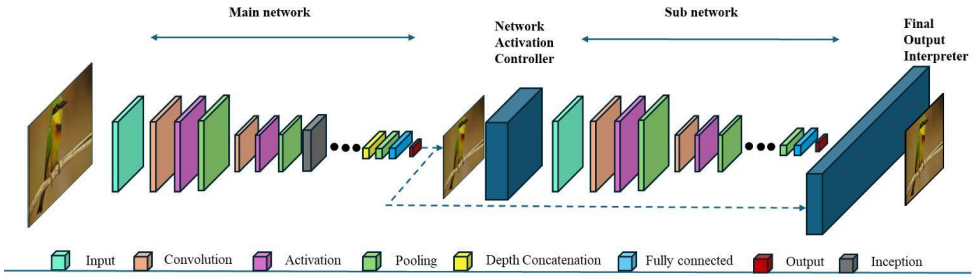


Fig. 3: Cascaded Architecture

3.1 Problem Formulation: CNN execution time reduction

In this section, the mathematical formulation of the problem statement is presented. Based on the data presented in Table 1, 2 and 3, it is clear that the number of layers of a convolutional neural network is increased with the intention of enhancing the classification accuracy of the model, which directly increases the number of complex computations involved and thus the overall execution time of the network also increases. From the literature presented, to achieve higher classification accuracy, a tendency of adapting more computational layers had been a common practice. However, it was interest in feed forward neural network that the classification accuracy be enhanced by incorporating more convolutions and more dense connections, even though it depends on various other factors such as training dataset, weights and bias distribution, stochastic drop-offs, information on the input data, etc. However, the presence of more convolutions results in more computations and thus in more inference time. We hypothesize that, by cascading convolutional neural networks with comparatively lesser learnable parameters, the overall execution time of the network model can be minimized.

3.2 Computations involved in CNN model

The layers of a CNN model can be broadly classified into two types, learnable layers and non-learnable layers. The learnable layers contain parameters (weights and biases) that are adjusted during the training process to minimize the loss function. Convolutional layers, fully connected layers, batch normalization layers, and activation layers are examples of learnable layers. However, pooling and dropout layers also involve simple mathematical operations, such as comparison. Considering the input image (I) as a two-dimensional data set where each element of the array represents the corresponding pixel value convoluted with a filter kernel (k) producing an output feature map (s) written in the general form of convolution.

$$s(p,q) = (k * I)(p,q) = \sum_{m} \sum_{n} I(p-m, q-n)k(m,n). \tag{1}$$

To calculate the number of computations involved in the convolutional layer, the size of the output feature map needed to be determined from a known input image. For an image input of size (H_{in}, W_{in}) where the input image height and width, respectively. Convolutional operation performed with the kernel of size (F_m, F_n) producing an output feature map of (H_{out}, W_{out}) with zero padding (P) over the stride of (S) in combination with dilation of (D). The height and width of the output feature map can be determined from following Eq.(2) and Eq.(3).

$$H_{out} = \left[\frac{H_{in} + 2 * padding[0] - dilation[0] * kernelsize[0] - 1}{stride[0]}\right] + 1. \tag{2}$$

$$W_{out} = \left[\frac{W_{in} + 2 * padding[1] - dilation[1] * kernelsize[1] - 1}{stride[1]}\right] + 1. \tag{3}$$

For an input image of size (W_{in}, H_{in}) with (P) color channels convoluted with a kernel of size (M, N) producing an output feature map of size (W_{out}, H_{out}) with depth of (K) require total computations denoted in the equation below.

$$No.of\ computations = P * M * (2N - 1) * W_{out} * H_{out} * K. \tag{4}$$

A feature map generated by convolutional layer is invalid unless passed through activation function. There are different activation functions used in neural networks such as ReLU, Sigmoid, tanh etc. However, ReLU is mostly used activation type among most network models. ReLU compares input value from every single neuron and replaces negative values with zero and positive values unchanged For a single neuron it requires one comparison and single multiplication to produce output to the next stage. If ReLU has (N) neurons, then

$$Total\ number\ of\ multiplications = N \times Computation\ for\ single\ neuron. \tag{5}$$

The mathematical operations involved are comparison and thresholding. Number of comparisons equivalent to the number of neurons (N). Thresholding operations don't involve additional calculations other than setting to zero. Therefore, the total number of calculations is equivalent to the number of neurons in the layer of interest. If the feature map is generated in a particular layer of size H x W x K (height, width, and depth of feature map) the total number of computations involved in ReLU layer is given by

$$No.of\ computations\ from\ ReLU = W_{out} * H_{out} * K. \tag{6}$$

To reduce the spatial dimension pooling functions are adapted in CNN models. The Max pooling was adapted For this research which selects the maximum value of the feature map generated per stride of the sliding window.For an input feature map of size $(W_{out}H_{out}P)$where width, height, and number of channels over a pooling filter of size(UV) width and height respectively with stride of (S) and depth of (K) the number of computations is determined by the number of pooling regions and number of comparisons within each pooling region. Number of pooling regions can be determined by

$$No.of\ regions = \left[\frac{(H_{out} - U)}{S} + 1\right] * \left[\frac{(W_{out} - V)}{S} + 1\right]. \tag{7}$$

The number of comparisons made within the pooling region is determined by the number of elements in the pooling filter as given in

$$No.of\ comparisons\ within\ the\ pooling\ region = U * V. \tag{8}$$

Therefore total number of computations in pooling layer can be derived from

$$No.of\ computations = \left[\frac{(H_{out} - U)}{S} + 1\right] * \left[\frac{(W_{out} - V)}{S} + 1\right] * U * V * K. \tag{9}$$

In convolutional neural network, as the input image is passed through the layers starting with input layer producing initial feature map, followed by sparse connections of convolutional layers should converge to specific neurons which ultimately represents one of the classes it is designated. Within the dense layer, each neuron is connected to every element of the input vector. Dot product is calculated between input vector and weights followed by addition of bias value. For input vector of size (J) with bias (b), the computations can be derived as the following equations

$$No.of\ computations\ per\ neuron = ((2J - 1) + b). \tag{10}$$

if I number of neurons are present at the fully connected layer, total number of computations from the fully connected layer can be obtained from

$$No.of\ computations\ from\ fully\ connected\ layer = I((2J - 1) + b). \tag{11}$$

3.3 Execution time measurement

As stated above, deep neural networks involve a larger number of mathematical computations in various layers of their architecture. The time taken by the neural network model to make a classification can be estimated with the computer involved for the classification process. It is noted that the execution time is a rough estimate but not a precise time measurement since the time is dependent on factors such as CPU clock frequency, CPU cores and Memory Bandwidth [33]. In convolutional layers of a network model, large number of multiplications and additions are repeated, which contribute to high latency as well as more power consumption [32]. Resource constraints in mobile computational platforms thus a significant challenge during the implementation of CNN models as computational complexity and energy efficiency should be critically considered. In addition to these challenges, since the CNN models involve a larger number of learnable parameters associated which requires large memory to store the instantaneous data [32][53]. Overseeing these constraints execution time estimation was conducted for the proposed cascaded network model and standard AlexNet. The AlexNet model was re-trained using transfer learning with the same dataset used for the cascaded network. For different object classes both the networks were trained using high performance computers and stored with respective class names for later experimental purposes. Time measurement was conducted on different computers to observe the characteristics of each CNN model. Test dataset consists of images used to

evaluate cascaded network and transfer learned AlexNet models to test the execution time taken by each network respectively. Each test image was taken as an input image to the network and executed the model to pass through all the layers until finally construct the class output. The execution time measurement was conducted for all the test images created in test dataset. Proposed cascaded network and AlexNet model were trained for different number of classes and stored individually to evaluated performances. Part of the execution time results for cascaded network are presented in Table 5. Each column represents the trained network model for different number of classes.

Table 5: Cascaded network's execution time for different classes

test object	15classes	25classes	40classes	50classes	100classes	200classes
cleaning machine	0.056511	0.05685	0.058375	0.057876	0.058957	0.060531
scanning machine1	0.059168	0.05937	0.059931	0.060483	0.062034	0.063023
scanning machine2	0.056495	0.056957	0.057696	0.058866	0.059594	0.060255
buggy cart	0.058392	0.05851	0.058583	0.05925	0.061399	0.061164
cat1	0.055661	0.055704	0.056601	0.058093	0.058458	0.059062
bird1	0.054368	0.055072	0.055469	0.055898	0.056488	0.058256
butterfly 3	0.058137	0.058461	0.058952	0.059811	0.060274	0.061036
dog31	0.061599	0.061649	0.061896	0.062368	0.063129	0.062002
pig5	0.059788	0.060251	0.060735	0.062368	0.062767	0.063893
pig1	0.059762	0.060034	0.06061	0.06113	0.061818	0.062713
pig14	0.05584	0.056087	0.056715	0.056966	0.057578	0.061265
buggy44	0.056159	0.056337	0.056727	0.056996	0.057801	0.059063
buggy36	0.064847	0.065471	0.065912	0.06659	0.067532	0.06897
stroller2	0.057536	0.057799	0.058371	0.058878	0.059553	0.062583
stroller3	0.058304	0.058654	0.059345	0.060833	0.061402	0.062253
trolley14	0.058017	0.058377	0.058664	0.059369	0.06026	0.061096
trolley20	0.05575	0.061071	0.06162	0.062176	0.062567	0.064581
Average time	0.0580196	0.058627	0.059188	0.059879	0.060683	0.061867

The same test data set was used to evaluate the re-trained AlexNet model. The AlexNet model trained on different number of output classes was represented in each column of Table 6 with execution time results.

3.3.1 Loss function

The classification accuracy of any convolutional neural network is a measure of how accurately the network model can predict the class label of a given input object of interest out of all objects that are learned to identify. Classification accuracy can be denoted as a mathematical formula.

$$Classification\ accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} * 100. \tag{12}$$

The proposed network model was evaluated for its classification accuracy to determine its performance as compared to other existing network models. Commonly used error matrices were employed for the accuracy analysis. Error distance (ED_i) is the quantitative measure which represents the arithmetic difference between the approximated

Table 6: Transfer learned AlexNet execution time for different classes

test object	15classes	25classes	40classes	50classes	100classes	200classes
cleaning machine	0.046271	0.046727	0.046825	0.048862	0.059822	0.067615
scanning machine1	0.044271	0.045267	0.047858	0.047858	0.058538	0.068017
scanning machine2	0.044641	0.045971	0.047769	0.048769	0.055598	0.066799
buggy cart	0.044491	0.046691	0.045232	0.046232	0.058611	0.067828
cat1	0.047282	0.045688	0.046283	0.049283	0.057879	0.067842
bird1	0.043623	0.045672	0.047216	0.049216	0.056779	0.065379
butterfly 3	0.043896	0.046077	0.047461	0.047461	0.058692	0.068572
dog31	0.046203	0.047541	0.048672	0.049662	0.056286	0.067015
pig5	0.046957	0.045911	0.047535	0.047975	0.058925	0.069996
pig14	0.045608	0.046628	0.047942	0.049342	0.055205	0.065876
buggy44	0.045366	0.046832	0.047954	0.048688	0.057669	0.066504
buggy36	0.045687	0.046793	0.045854	0.047863	0.055624	0.066221
stroller2	0.045441	0.045911	0.047165	0.048261	0.055941	0.065783
stroller3	0.046524	0.046631	0.047547	0.049547	0.058215	0.067976
trolley14	0.046633	0.047438	0.047137	0.047911	0.059405	0.068392
trolley14	0.045842	0.046164	0.046915	0.048527	0.057822	0.067615
average time	0.045546	0.046371	0.04721	0.048466	0.057563	0.067339

result (R_i') and the actual result(R_i) mathematically represented in

$$ED_i = |R_i' - R_i|. \tag{13}$$

Mean Error Distance (MED) is the average error distance for N number of test trials can be mathematically represented in

$$MED = \frac{1}{N}\sum_{i=1}^{N} ED_i. \tag{14}$$

Considering the relative error distance (RED_i), which is a measure of error distance (ED) with respect to the actual result (R) how much deviated from the actual result. Relative error distance can be mathematically denoted in

$$RED_i = \left|\frac{(R_i' - R_i)}{R_i}\right|. \tag{15}$$

Mean Relative Error Distance (MRED) is the average relative error distance for N number of test trials which can be mathematically represented in

$$MRED = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{(R_i' - R_i)}{R_i}\right|. \tag{16}$$

Mean Square Error (MSE) is the average of the squared error distances, which can be mathematically denoted in

$$MSE = \frac{1}{N}\sum_{i=1}^{N}|R_i' - R_i|^2. \tag{17}$$

3.4 Inference process

Trained cascaded network and transfer-learned AlexNet model was tested on test dataset derived from original ImageNet dataset, which was not used for either training or validation process. During the inference stage, an image from the test dataset was passed through the network model as an input image. Main network block compute the object class confidence and estimator decides the second stage network to be invoked. Second stage uses the primary feature map generated for its final prediction of the output class. Once both output classes determined returns the predicted output. Inference time is measured at the beginning of the execution of the cascaded network is used to determine the total inference time upon successful completion of final class estimation. Algorithm 1 presents the classification process involved in cascaded neural network model.

Algorithm 1 Test for the cascaded network

Input: x - test dataset, pre-trained model, N - iterations
Output: classification accuracy, average execution time

Import required libraries
Load pre-trained AlexNet model
Pre-process:
resize the input data to image size of 227 x 227 pixels with 3 channels
input_data = resize(x, [227 227])
Classification confidence:
probabilities = model.softmax(output)
top_confidence = max(probabilities)
Measure average execution time:
Initialize cumulative_time = 0
for $i = 1 : N$ **do**
    Start_time = current_time()
    Perform forward pass: model(input_data)
    End_time = current_time()
    update cumulative_time += (end_time - start_time)
end for
Average_time = cumulative_time / N
Output: print top_confidence, average_time
END

4 Experiment

4.1 Implementation details

The Proposed system was implemented on MATLAB using Deep Learning Toolbox. The network architecture was analyzed with the built-in analyzing tool.The proposed cascaded network was designed to be implemented on a passenger guidance service robot in an airport environment. The mobile service robot equipped with limited computational resources and power constraints should work independently as an EDGE device. These design constraints led us to research energy efficient, accurate convolutional neural network design which could efficiently work on mobile computation platform. Since a very limited number of objects are available within an airport environment, the proposed cascaded network was trained only for a maximum of 200 object classes. The data set extracted from ImageNet ILSVRC 2012 was used for training and validation in 7:3 ratio over 200 classes. For the bench marking purpose standard AlexNet was transfer learned with the same dataset used for the cascaded network architecture. Test dataset was created from the same ImageNet dataset. The inference time measurement is essential in time critical real-world applications such as active safety systems in automobiles, product sorting in high speed process control etc. For applications which requires low latency, it is necessary that the network model produces accurate predictions in a limited time. Inference time was measured in two methods such as computer timing library and MATLAB profiler tool. This entire process can be explained in five steps.

1. Select the input: The input data sample or batch of samples to be classified. It is necessary that the input samples were not used during the network model training process, and it should be real world data.
2. Start timing: record the start time before feeding the input sample to the pretrained neural network.
3. Execute forward pass: pass the input data through the neural network's all the layers starting the image input layer up to classification output layer where the classification prediction is obtained.
4. Stop timing: record the end time once the forward pass is completed and the classification output is produced.
5. Calculate the inference time and accuracy: The time elapsed or the difference between stop time and start time gives the inference time.

4.2 Results for cascaded network

Referring to the execution time comparison presented in Fig. 4, it was observed that as the number of classes were increased, the execution time for the Alexnet increased gradually with a lower rate of increment initially for lower output classes, which then increased over the number of classes. Cascaded network on the other hand recorded comparatively higher execution time of 58ms at the beginning for low output classes which is 13ms higher than Alexnet. It was observed that cascaded network execution time increases over the number of output classes. However, the rate at which it increases is less as compared to Alexnet. It was also observed that the execution time

of both networks was equal to approximately 135 classes. Similar approach presented in [54] that highlights the effectiveness of accuracy and cascading architecture. The computer on which we conducted the test has a base frequency of 2.4 GHz. If the process runs at the base frequency, the single machine cycle time is $4.1666x10^{-}10$ seconds. Since a single instruction execution requires 4 machine cycles, the time to execute one computation is $1.041666x10^{-}10$ seconds. Since there are 4 cores in the processor, the execution time is $2.60416668x10^{-}11$ seconds. To execute the cascaded network with 25 classes, there are 2556548992 computations to be performed. Hence, the maximum possible time required to execute is 0.066576797 seconds. This is within the practical execution time observed during the execution of the network model

4.3 Qualitative results

The performance of the proposed cascaded network model was tested using a test dataset derived from the original ImageNet ILSVRC dataset, ensuring that the images used for the test dataset were not included in the training or validation process. The same test data was applied to AlexNet, and execution time and classification accuracy were recorded for further analysis. The graph in Fig. 4 illustrates how the execution time varies with the number of output classes.

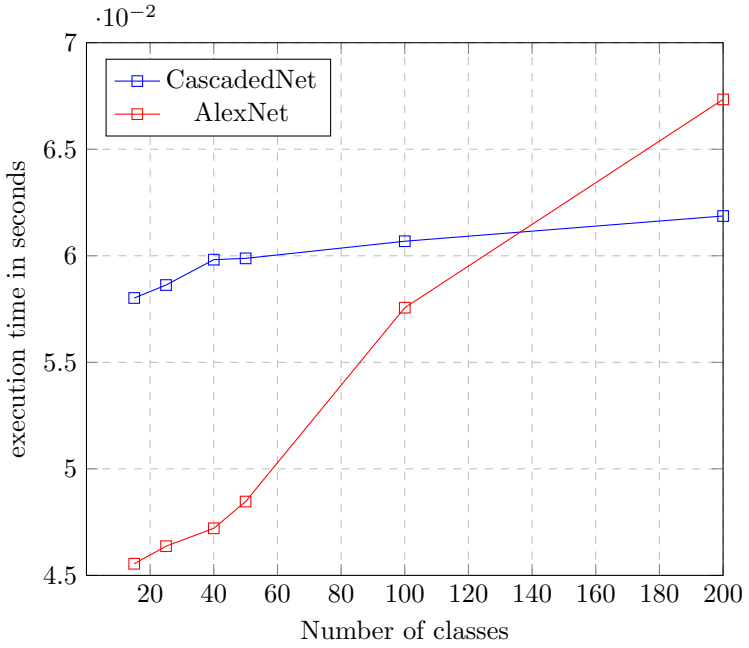


Fig. 4: Comparison of model execution time

Since the research was focused on limited object environment (airport application), it was tested only for a maximum of 200 object classes due to it's intended application constraints. However, a general observation of this research was as the number of output classes increases, the execution time also increases. The rate at which the execution time increases is less in the cascaded approach as compared to the existing standard CNN model. The performance of the other network models was also considered. The datasets used for the proposed network were used to retrain the other network models such as ResNet-50, GoogleNet, DenseNet-201 and EfficientNet. The graph plotted in Fig 5 presents the execution time variation over the increment of number of classes. Training individual network models was a time-consuming task,

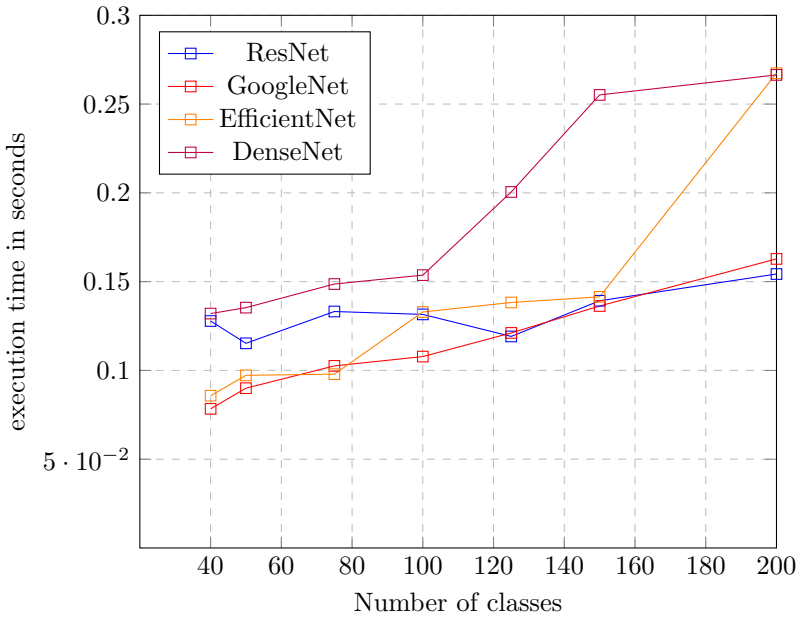


Fig. 5: Comparison of other CNN models execution time

particularly for complex architectures such as DenseNet and EfficientNet. After the models were transfer-learned, they were individually evaluated on a test dataset consisting of approximately 1000 sample images. Referring to the classification accuracy of the cascaded network, presented in Fig. 6 for different image inputs. The error bar chart represents the percentage classification accuracy along with highest and lowest recorded values. From the data presented in Fig. 6 it was observed that the cascaded network model offers 95% classification accuracy with an average lowest classification of 75%. While the cascaded network offers better classification results, the lowest recorded average value of 93% with the lowest confidence value of 54% was observed from one of the classes out of the test data input. Similarly, the transfer-learned Alexnet from the transfer learning was tested on the same object categories using the same test images prepared to be tested over the proposed network model. The

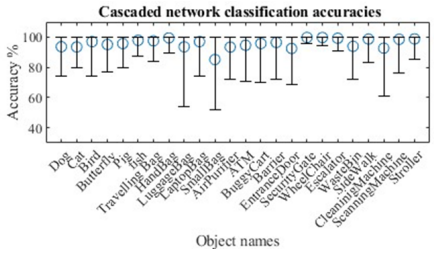


Fig. 6: Cascaded network classification accuracies

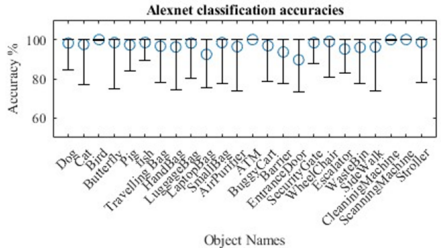


Fig. 7: Alexnet classification accuracies

results were tabulated and presented on Fig. 7. an average classification accuracy of 97% was observed with an average lowest classification accuracy of 83%. The recorded least accuracy of 75% for one object class could be observed while outperforming the most cases. On comparison it can be observed that the proposed cascaded network model's classification accuracy is slightly low as 2% compared with Alexnet network model. All in all, the proposed cascaded network model is satisfactorily accurate while producing its prediction in the shortest time. The proposed cascaded network uses convolutional layers to extract the features from input data. These features are passed through the fully connected layer to predict the label of input data. Regarding the unseen data, the performance of the cascaded network model may vary due to several reasons. If the unseen data is in domain similar to the trained dataset, the cascaded network perform well. However, if the unseen data is from a different domain such as thermal image, satellite image or medical image the performance of the cascaded network drops unless transfer learning is applied. Since the proposed cascaded network model is simple architecture as compared to the other models such as Dense net, the performance may be degraded for highly complex unseen data

5 Future work

The potential future research can also be presented based on the developments of the research work. Further analysis of the results obtained, a tendency of minimum learnable parameters can be observed for the selection of number of classes to the main and sub network. The potential combinations for the classes were centered to minimum values possible. Thus, cascading of third and higher order convolutional

neural networks are a potential feature where the present two stages can further be extended. Development of hierarchical dataset is essential to facilitate more meaningful and human-like classification. The proposed model consists of two stage cascaded network for classification applications with lesser number of classes. This concept can be further expanded by cascading a greater number of optimized networks together for a large number of classes. The optimum number of cascades can be determined by conducting practical execution time measurement over the number of classes. Future work is required to investigate the performance of multi-stage cascaded neural network. The neural network model presented with this research work consists of modified Alex net architecture. Convolution operation of an intermediate layer depends on feature map developed from the previous layer convoluted with input feature map followed by an activation function. Therefore, convolution operation towards the dense layers is dependent on the features extracted from prior layers. Instead of single feature extraction, if multiple features could be extracted concurrently and concatenate them at the last convolutional layers possibly improve classification accuracy in considerably less time. Further investigation of the architectural modification with deep training and analysis of results can result in more accurate and efficient models which can be cascaded Further development of this cascaded approach has the potential to deploy on mobile robotic systems with standalone computer platforms which work offline, function efficiently. However, a deep analysis of the network is essentially supported with mathematical representation.

6 Conclusion

From the results achieved, the cascaded network offered reduced execution time as compared with Alex Net over the increasing number of output classes. The proposed cascaded network model is 6.43% faster than the standard AlexNet model with the same number of output classes. Comparing classification accuracies, the cascaded network model is 2% less confident to the equivalent of standard AlexNet.The accuracy reduction is a result of several factors. The classification accuracy depends on the information of the input image, classification model, quantity and quality of training data, distribution of training data etc. For this research work 30% of the dataset was allocated for validation purposes while collection of test data consists of 1000 sample images were taken for cascaded network model evaluation. To further improve classification accuracy, training the network model with more data, dataset with balanced information content, training the model with slow learning rate are suggested. However, the accuracy of the cascaded model can be further improved with the suggestions presented in future work. The primary contribution of this research was to reduce the complex computations involved in convolutional neural networks. Modifications introduced to the neural network architecture in terms of layer-wise changes have been a significant factor for reduction of unnecessary computations as compared with standard neural models. As a result of the modifications introduced to the network model, a considerably large reduction of learnable parameters was observed which reflects the effective computation optimizations compared with other network models such as GoogleNet. Architectural modifications introduced with this research is the second

contribution which includes convolution with varying kernel sizes from, batch normalization, concurrent feature extractions etc. Due to the modifications introduced to the architecture resulted in minimizing of computations and comparisons in both sparse and dense layers. Since the number of complex computations are reduced, the time taken to execute the network also reduced and thus the inference time. Parallel feature extraction towards the last convolutional layers has significantly reduced the execution time of the network model and thereby speeding up the classification process. Adaptation of computationally efficient functions such as rectified and linear units for the feature activation instead of parametric rectified linear unit, tanh, or exponential linear units. It not only overcomes unnecessary computations but also minimizes the time to compute exponents. The third contribution of this work is the proposed network model that maintains better classification accuracy while optimizing the computations and reducing inference time. Normalization techniques introduced after every feature activation has resulted in better classification capability of the proposed network model.

Statements and Declarations

- **Data Availability Statement** The data used in this study are available from third-party institutions. Researchers can access the data through the instructions presented in the original work of the corresponding datasets. ImageNet (ILSVRC2012) Dataset available at https://www.image-net.org/challenges/LSVRC/2012/index.php. Test data, codes and neural networks are available at https://github.com/isuru8787/CascadedNet-Page.
- **Funding** No funding was received to assist with the preparation of this manuscript.
- **Conflict of Interest** The Authors have no competing interests to declare that are relevant to the content of this article.

References

[1] S. Durvasula, Y.G., Vijaykumar, N.: Ev-conv: Fast cnn inference on event camera inputs for high-speed robot perception. IEEE Robot Autom Lett **8**(6), 3174–3181 (2023)

[2] Phuong, T.M., Diep, N.N.: Speeding Up Convolutional Object Detection for Traffic Surveillance Videos. 2018 10th International Conference on Knowledge and Systems Engineering (KSE), Ho Chi Minh City, Vietnam (2018)

[3] S. Zhai, D.S. S. Dong, Wang., S.: An improved faster r-cnn pedestrian detection algorithm based on feature fusion and context analysis. IEEE Access **8**, 138117–138128 (2020)

[4] Fukushima., K.: Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics **36**(4), 193–202 (1980)

[5] K. He, S.R. X. Zhang, Sun, J.: Deep Residual Learning for Image Recognition. http://arxiv.org/abs/1512.03385. Accessed: 2023-05-30

[6] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017) https://doi.org/10.1145/3065386

[7] Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998) https://doi.org/10.1109/5.726791

[8] Zeiler, M.D., Fergus, R.: Visualizing and Understanding Convolutional Networks. http://arxiv.org/abs/1311.2901. Accessed: 2023-05-30

[9] Redmon, J., Farhadi, A.: Yolo9000: Better, faster, stronger. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6517–6525 (2017). https://doi.org/10.1109/CVPR.2017.690

[10] Nie, Y., Sommella, P., O'nils, M., Liguori, C., Lundgren, J.: Automatic detection of melanoma with yolo deep convolutional neural networks. 2019 E-Health and Bioengineering Conference (EHB), 1–4 (2019)

[11] Lavin, A., Gray, S.: Fast algorithms for convolutional neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4013–4021 (2016). https://doi.org/10.1109/CVPR.2016.435

[12] Vasilache, N., Johnson, J., Mathieu, M., Chintala, S., Piantino, S., LeCun, Y.: Fast Convolutional Nets With fbfft: A GPU Performance Evaluation (2015). https://arxiv.org/abs/1412.7580

[13] Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X.: Pruning and quantization for deep neural network acceleration: A survey. Neurocomput. **461**(C), 370–403 (2021) https://doi.org/10.1016/j.neucom.2021.07.045

[14] Ren, P., Xiao, Y., Chang, X., Huang, P., Li, Z., Chen, X., Wang, X.: A comprehensive survey of neural architecture search: Challenges and solutions. CoRR **abs/2006.02903** (2020) 2006.02903

[15] Cireşan, D., Meier, U.: Multi-column deep neural networks for offline handwritten chinese character classification. In: 2015 International Joint Conference on Neural Networks (IJCNN), pp. 1–6 (2015). https://doi.org/10.1109/IJCNN.2015.7280516

[16] Cireşan, D.C., Meier, U., Masci, J., Gambardella, L.M., Schmidhuber, J.: High-Performance Neural Networks for Visual Object Classification (2011). https://arxiv.org/abs/1102.0183

[17] Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis. In: Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings., pp. 958–963 (2003). https://doi.org/10.1109/ICDAR.2003.1227801

[18] Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition (2015). https://arxiv.org/abs/1409.1556

[19] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9 (2015). https://doi.org/10.1109/CVPR.2015.7298594

[20] Lin, R.: Analysis on the selection of the appropriate batch size in cnn neural network. 2022 International Conference on Machine Learning and Knowledge Engineering (MLKE), 106–109 (2022)

[21] Saeed, R., Ghnemat, R., Benbrahim, G., Elhassan, A.: Learning with dynamic architectures for artificial neural networks - adaptive batch size approach. In: 2019 2nd International Conference on New Trends in Computing Sciences (ICTCS), pp. 1–4 (2019). https://doi.org/10.1109/ICTCS.2019.8923070

[22] Ahn, J.Y., Cho, N.I.: Multi-branch neural architecture search for lightweight image super-resolution. IEEE Access **9**, 153633–153646 (2021) https://doi.org/10.1109/ACCESS.2021.3127437

[23] Yu, Y., Jiang, J., Chi, X.: Using supercomputer to speed up neural network training. In: 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), pp. 942–947 (2016). https://doi.org/10.1109/ICPADS.2016.0126

[24] Brillet, L.F., Leclaire, N., Mancini, S., Cleyet-Merle, S., Nicolas, M., Henriques, J.P., Delnondedieu, C.: Speeding-up cnn inference through dimensionality reduction. In: 2019 Conference on Design and Architectures for Signal and Image Processing (DASIP), pp. 1–6 (2019). https://doi.org/10.1109/DASIP48288.2019.9049204

[25] Guo, C., Hsueh, B.Y., Leng, J., Qiu, Y., Guan, Y., Wang, Z., Jia, X., Li, X., Guo, M., Zhu, Y.: Accelerating Sparse DNN Models without Hardware-Support via Tile-Wise Sparsity (2020). https://arxiv.org/abs/2008.13006

[26] Yang, S., He, S., Duan, H., Chen, W., Zhang, X., Wu, T., Yin, Y.: Apq: Automated dnn pruning and quantization for reram-based accelerators. IEEE Transactions on Parallel and Distributed Systems **34**(9), 2498–2511 (2023) https://doi.org/10.1109/TPDS.2023.3290010

[27] Fukagai, T., Maeda, K., Tanabe, S., Shirahata, K., Tomita, Y., Ike, A., Nakagawa,

A.: Speed-up of object detection neural network with gpu. In: 2018 25th IEEE International Conference on Image Processing (ICIP), pp. 301–305 (2018). https://doi.org/10.1109/ICIP.2018.8451814

[28] Pasquet, J., Chaumont, M., Subsol, G., Derras, M.: Speeding-up a convolutional neural network by connecting an svm network. In: 2016 IEEE International Conference on Image Processing (ICIP), pp. 2286–2290 (2016). https://doi.org/10.1109/ICIP.2016.7532766

[29] Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence **39**(6), 1137–1149 (2017) https://doi.org/10.1109/TPAMI.2016.2577031

[30] Pang, J., Li, C., Shi, J., Xu, Z., Feng, H.: $\mathcal{R}^2$ -cnn: Fast tiny object detection in large-scale remote sensing images. IEEE Transactions on Geoscience and Remote Sensing **57**(8), 5512–5524 (2019) https://doi.org/10.1109/TGRS.2019.2899955

[31] Zhang, L.L., Han, S., Wei, J., Zheng, N., Cao, T., Yang, Y., Liu, Y.: nn-meter: towards accurate latency prediction of deep-learning model inference on diverse edge devices. In: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services. MobiSys '21, pp. 81–93. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3458864.3467882 . https://doi.org/10.1145/3458864.3467882

[32] Hazarika, A., Poddar, S., Nasralla, M.M., Rahaman, H.: Area and energy efficient shift and accumulator unit for object detection in iot applications. Alexandria Engineering Journal **61**(1), 795–809 (2022) https://doi.org/10.1016/j.aej.2021.04.099

[33] Green, S., Vineyard, C.M., Koç, Ç.K.: In: Koç, Ç.K. (ed.) Mathematical Optimizations for Deep Learning, pp. 69–92. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98935-8_4 . https://doi.org/10.1007/978-3-319-98935-8$_4$

[34] Chang, J., Choi, Y., Lee, T., Cho, J.: Reducing mac operation in convolutional neural network with sign prediction. In: 2018 International Conference on Information and Communication Technology Convergence (ICTC), pp. 177–182 (2018). https://doi.org/10.1109/ICTC.2018.8539530

[35] Moini, S., Alizadeh, B., Emad, M., Ebrahimpour, R.: A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications. IEEE Transactions on Circuits and Systems II: Express Briefs **64**(10), 1217–1221 (2017) https://doi.org/10.1109/TCSII.2017.2690919

[36] Ahmed, H.O., Ghoneima, M., Dessouky, M.: Concurrent mac unit design using vhdl for deep learning networks on fpga. 2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE), 31–36 (2018)

[37] Ma, J., Salcic, Z.: Deep learning with accelerated execution: A real-time license plate localisation system. In: 2021 IEEE 19th International Conference on Industrial Informatics (INDIN), pp. 1–8 (2021). https://doi.org/10.1109/INDIN45523.2021.9557474

[38] Xiao, H., Xu, H., Chen, X., Wang, Y., Han, Y.: Fast and high-accuracy approximate mac unit design for cnn computing. IEEE Embedded Systems Letters **14**(3), 155–158 (2022) https://doi.org/10.1109/LES.2021.3137335

[39] Anbananthen, K., S, S., Baskar, S., Selvaraj, R., Krishnan, J., Kannan, S., Chelliah, D.: The eye: A light weight mobile application for visually challenged people using improved yolov5l algorithm. Emerging Science Journal **7**, 1636–1652 (2023) https://doi.org/10.28991/ESJ-2023-07-05-011

[40] Surono, S., Firza Afitian, M.Y., Setyawan, A., Arofah, D., Thobirin, A.: Comparison of cnn classification model using machine learning with bayesian optimizer. HighTech and Innovation Journal **4**, 531–542 (2023) https://doi.org/10.28991/HIJ-2023-04-03-05

[41] Song, P., Chen, K., Zhu, L., Yang, M., Ji, C., Xiao, A., Jia, H., Zhang, J., Yang, W.: An improved cascade r-cnn and rgb-d camera-based method for dynamic cotton top bud recognition and localization in the field. Comput. Electron. Agric. **202**(C) (2022) https://doi.org/10.1016/j.compag.2022.107442

[42] Wu, Y., Liu, W., Wan, S.: Multiple attention encoded cascade r-cnn for scene text detection. Journal of Visual Communication and Image Representation **80**, 103261 (2021) https://doi.org/10.1016/j.jvcir.2021.103261

[43] Wang, Z., Lv, J., Yang, Y., Lin, Y., Li, Q., Li, X., Yang, X.: Accurate scoliosis vertebral landmark localization on x-ray images via shape-constrained multi-stage cascaded cnns. Fundamental Research (2022) https://doi.org/10.1016/j.fmre.2022.10.014

[44] Wang, K., Dong, Y., Bai, H., Zhao, Y., Hu, K.: Use fast r-cnn and cascade structure for face detection. In: 2016 Visual Communications and Image Processing (VCIP), pp. 1–4 (2016). https://doi.org/10.1109/VCIP.2016.7805472

[45] Zhao, J., Zhu, H., Liu, B.: Deep learning: A study of pattern recognition for personalized clothing. HighTech and Innovation Journal **4**, 505–514 (2023) https://doi.org/10.28991/HIJ-2023-04-03-03

[46] Thilagavathi, M., Abirami, S.: Cascade-forward neural network in identification of plant species of desert based on wild flowers. In: 2018 IEEE International Conference on System, Computation, Automation and Networking (ICSCA), pp. 1–7 (2018). https://doi.org/10.1109/ICSCAN.2018.8541172

[47] Al-Nima, R.R.O., Abdulraheem, F.H., Al-Ridha, M.Y.: Using hand-dorsal images

to reproduce face images by applying back propagation and cascade-forward neural networks. In: 2019 2nd International Conference on Electrical, Communication, Computer, Power and Control Engineering (ICECCPCE), pp. 97–102 (2019). https://doi.org/10.1109/ICECCPCE46549.2019.203755

[48] Li, Y., Wang, Z.: Research on textile defect detection based on improved cascade r-cnn. In: 2021 International Conference on Artificial Intelligence and Electromechanical Automation (AIEA), pp. 43–46 (2021). https://doi.org/10.1109/AIEA53260.2021.00017

[49] Cai, Z., Vasconcelos, N.: Cascade r-cnn: High quality object detection and instance segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence **43**(5), 1483–1498 (2021) https://doi.org/10.1109/TPAMI.2019.2956516

[50] Dong, H., Li, Y., Liu, R.: A detection algorithm based on improved cascade r-cnn for uav aerial images. In: 2023 IEEE 3rd International Conference on Electronic Technology, Communication and Information (ICETCI), pp. 700–704 (2023). https://doi.org/10.1109/ICETCI57876.2023.10176806

[51] Hurtik, P., Ozana, S.: Dragonflies segmentation with u-net based on cascaded resnext cells. In: Neural Computing and Applications, vol. 33, pp. 4567–4578 (2021). https://doi.org/10.1007/s00521-020-05274-y

[52] Chen, T., Hu, X., Xiao, J., Zhang, G., Wang, S.: Cfidnet: cascaded feature interaction decoder for rgb-d salient object detection. In: Neural Computing and Applications, vol. 34, pp. 7547–7563 (2022). https://doi.org/10.1007/s00521-021-06845-3

[53] Chang, J., Choi, Y., Lee, T., Cho, J.: Reducing mac operation in convolutional neural network with sign prediction. In: 2018 International Conference on Information and Communication Technology Convergence (ICTC), pp. 177–182 (2018). https://doi.org/10.1109/ICTC.2018.8539530

[54] Zou, W., Luo, Y., Cao, W., He, Z., He, Z.: A cascaded registration network rcinet with segmentation mask. In: Neural Computing and Applications, vol. 33, pp. 16471–16487 (2021). https://doi.org/10.1007/s00521-021-06243-9

End of Thesis