**UNIVERSITI TEKNOLOGI MARA**

# FPGA IMPLEMENTATION OF A CO-DESIGNED CNN AND FILTERING BLOCK FOR DIGIT SIGN LANGUAGE RECOGNITION

**JOEL LAING LUYOH**

**MSc**

**June 2026**

**UNIVERSITI TEKNOLOGI MARA**

# FPGA IMPLEMENTATION OF A CO-DESIGNED CNN AND FILTERING BLOCK FOR DIGIT SIGN LANGUAGE RECOGNITION

**JOEL LAING LUYOH**

Thesis submitted in fulfilment
of the requirements for the degree of
**Master of Science**
**(Electrical Engineering)**

**Faculty of Electrical Engineering**

**June 2026**

# CONFIRMATION BY PANEL OF EXAMINERS

I certify that a Panel of Examiners has met on 8 May 2026 to conduct the final examination of Joel Laing Luyoh on his Masters of Science thesis entitled "FPGA Implementation of a Co-Designed CNN and Filtering Block for Digit Sign Language Recognition" in accordance with Universiti Teknologi MARA Act 1976 (Akta 173). The Panel of Examiner recommends that the student be awarded the relevant degree. The Panel of Examiners was as follows:

Alhan Farhanah Abd Rahim, PhD
Associate Professor
Faculty of Electrical Engineering
Universiti Teknologi MARA
(Chairman)

Irni Hamiza Hamzah, PhD
Associate Professor
Faculty of Electrical Engineering
Universiti Teknologi MARA
(Internal Examiner)

Khairol Amali Ahmad, PhD
Professor
University Pertahanan Nasional
Malaysia
(External Examiner)

**PROFESSOR DR HJH ZURAEDA IBRAHIM**
Dean
Institute of Postgraduates Studies
Universiti Teknologi MARA

Date: 24 June 2026

# AUTHOR'S DECLARATION

I declare that the work in this thesis was carried out in accordance with the regulations of Universiti Teknologi MARA. It is original and is the results of my own work, unless otherwise indicated or acknowledged as referenced work. This thesis has not been submitted to any other academic institution or non-academic institution for any degree or qualification.

I, hereby, acknowledge that I have been supplied with the Academic Rules and Regulations for Postgraduate, Universiti Teknologi MARA, regulating the conduct of my study and research.

| | | |
|---|---|---|
| Name of Student | : | Joel Laing Luyoh |
| Student ID. No. | : | 2024831926 |
| Programme | : | Master of Science (Electrical Engineering) – CEEE750 |
| Faculty | : | Electrical Engineering |
| Thesis Title | : | FPGA Implementation of a Co-Designed CNN and Filtering Block for Digit Sign Language Recognition |
| Signature of Student | : | …….. |
| Date | : | June 2026 |

# ABSTRACT

Sign language (SL) recognition system using convolutional neural network (CNN) have shown high accuracy performance in software level but its implementation on a portable hardware remains a challenge due to CNN's computational demands and the lack consideration of hardware constraints during development. Moreover, the challenge is also due to the complex architecture of CNN that requires hardware with large memory resources and attain high computational cost. In addition, real-time image capturing introduces noises that necessitates the need of pre-processing filters. However, similar to the challenges of CNN, the filters implementation is also typically designed only in software without the hardware consideration. Therefore, this research aims to design and develop a functional CNN-based digit SL classification model that can implemented on field-programmable gate array (FPGA) hardware and enhanced the performance by a co-design approach of incorporating pre-processing filter block to the system. The feasibility of the CNN models with co-design filters are evaluated for real-time classification. In achieving the objectives of the research, a 7-layer CNN architecture was developed in both matrix laboratory (MATLAB) and Quartus software, then trained in MATLAB using an American digit SL dataset. Four filters (gaussian, median, bilateral and sobel) were first implemented in the FPGA pipeline, then adapted the design in MATLAB for training following the co-design approach. The weights and biases of the trained models were converted to the compatibility of FPGA format and synthesized with the designed CNN system in the FPGA pipeline. The CNN is implemented on the Terasic DE2-115 FPGA development board with D8M-GPIO camera for real-time testing. The benchmark CNN model without filter achieved 92.43% training accuracy, 88.53% validation accuracy, 69.33% testing accuracy in MATLAB and 14.33% real-time testing accuracy on FPGA using 90,612 logic elements (LEs). The bilateral filter model scores the best relative improvements over the benchmark model with +12.32% in MATLAB testing accuracy and +20.94% in real-time testing accuracy on FPGA. The performance hierarchy followed by the median filter model with improvement of +11.35% in MATLAB testing accuracy and +18.14% in real-time testing accuracy on FPGA, while gaussian filter model relatively improves by +6.16% in MATLAB testing accuracy and +8.86% in real-time testing accuracy on FPGA. The sobel filter model failed completely with only 9.97% training accuracy. In hardware implementation, all successful filter models utilized 95% to 97% of the available resources in FPGA. Looking at the consistent performance hierarchy (bilateral > median > gaussian > without filter) in both MATLAB and FPGA platforms, this validates the co-design approach in CNN. This research has successfully demonstrated that the edge-preserving filter, bilateral filter in particular, implemented through the co-design approach enhance the CNN performance for digit SL classification on FPGA significantly. The implementation of FPGA provides a realization of future application-specific integrated circuit (ASIC) development, producing practical and portable communication aided device to a reality for the deaf or hard of hearing community.

# ACKNOWLEDGEMENT

First of all, all glory to God and I am forever grateful to Jesus for the open door and unmerited favor in embarking my master's and leading me in completing this long and challenging journey well.

My gratitude and appreciation go to my supervisor, Assoc. Prof. Ir. Ts. Dr. Samsul Setumin for the wonderful supervision and guidance from the beginning and all the way to the end. Also, I give huge thanks to my co-supervisors, Ir. Dr. -Ing. Emilia Noorsal and Dr. Siti Juliana Abu Bakar for the stern and constructive feedback as well as comments that helps in firming my direction of research. Special thanks to my colleagues and friends within the Machine Learning Research Group (MLRG) and College of Engineering for helping me in this journey.

This thesis is dedicated to my loving parents and family afar back home in Sarawak for the endless encouragement and prayers to keep me walk in obedience in this journey. This achievement is dedicated to all of you. I want to mention also my spiritual family here in EPCC Summit. Thank you for the prayers, uplifting words and reminders of His Words. I also thanked my love, Bel, that constantly being patience and loving me through the ups and downs as well as the shoulder lend when it's tough. I love you dear.

Finally, I would say thank you to the myself for being resilient and dedicated through it all. Good job and well done. Halleluyah!

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF SYMBOLS

## Symbols

| | |
|---|---|
| i | i-th sample |
| i , j | Offset indices ranging from -1 to 1 |
| I(x,y) | Input image pixel value at coordinate (x,y) |
| $L_C$ | Cross-entropy loss |
| $L_Q$ | Quadratic loss |
| $M_{filter}$ | Metric value for the CNN model with filters |
| $M_{benchmark}$ | Metric value for the benchmark CNN model without filter |
| N | Number of samples in a batch |
| $p_i$ | Predicted probability of class 1 |
| $R_{filter}$ | Resource utilization for the CNN model with filters |
| $R_{benchmark}$ | Resource utilization for the benchmark CNN |
| $w_s$ | Spatial weight based on distance from centre pixel |
| $w_r$ | Range weight based on intensity difference |
| W | Normalization factor |
| $\hat{y}_i$ | Predicted output for sample *i* |
| $y_i$ | True label for sample *i* |
| (x,y) | Output pixel value at coordinate (x,y) |

# LIST OF ABBREVIATIONS

**Abbreviations**

| | |
|---|---|
| ARM | Advanced (RISC) Machine |
| ASIC | Application-specific integrated circuit |
| BNN | Binarized neural network |
| BRAM | Block Random Access Memory |
| CNN | Convolutional neural network |
| CPU | Central Processing Unit |
| DHH | Deaf and hard of hearing |
| DSP | Digital Signal Processing |
| FC | Fully Connected |
| FPGA | Field-programmable gate array |
| fps | Frame per Second |
| GPU | Graphic Processing Unit |
| LE | Logic Element |
| MAC | Multiply-Accumulate |
| RAM | Random Access Memory |
| RISC | Reduced Instruction Set Computer |
| SL | Sign Language |

# CHAPTER 1
# INTRODUCTION

## 1.1 Research Background

Sign language (SL) is a visual language that uses hand gestures as a means of communication for the deaf or hard of hearing (DHH) community [1], [2]. SL is a method whereas the DHH people express their thoughts and ideas using hand gestures and signs without verbalizing to others. Today, the DHH community is still at the disadvantages due to communication gap in the society [3]–[6]. With the advanced work of technology, an automated sign language recognition and classification system with applications like assistive technology and computer vision have become a critical research area to bridge the gap.

Convolutional neural network (CNN) has completely changed the field of machine learning, especially in executing tasks that mitigate human visual such as object classification and recognition [7]. CNN is a type of deep learning model that consists of a series of layers that operate to extract hierarchical features from raw visual data automatically with high efficiency [8]. Therefore, CNN's ability is highly suitable for sign language recognition and classification, where hand gestures and signs are complex and varies across different countries.

Among the CNN models, classification models are known for its effectiveness in categorizing visual data into distinct classes. CNN classification models are designed particularly to analyse input images and extract key features to provide accurate predictions of classification [9]. These models have proved their effectiveness in recognizing a wide range of sign language gestures such as hand shapes and movements. Through training on large datasets of SL images, CNN classification model is able to classify hand signs and gestures.

CNN classification models can be implemented for a real-time on a hardware like Field-Programmable Gate Array (FPGA). FPGA is a platform commonly known of its parallel computation performance, fast processing speed and low power consumption. This makes it suitable for the implementation of intensive computation of CNN [10], [11]. This also makes FPGA viable for a real-time SL classification as it offers advantages in terms of efficiency, speed, flexibility and scalability. Moreover,

enabling a portable and energy-efficient prototype solution in application-specific integrated circuit (ASIC) application for various implementation such as smart environments and communications aid.

The aim of this study is developing a CNN digit SL classification model prototype on FPGA which is feasible for SL classification. The research focuses on training, validating and testing the CNN model and its implementation on the FPGA platform. By addressing the objectives of the project, the end goal is bridging the gap of communication for the deaf-mute community with a feasible prototype for SL classification.

## 1.2 Motivation for This Work

The main motivation of this research is to produce a practical and accessible technology for the deaf or hard of hearing (DHH) community for daily communication with the society. The lack of sign language (SL) proficiency in the society creates the communication barriers to education, healthcare and social inclusion. With the development of the right device, it raises confidence for DHH community to have the sense of belonging in the society and inclusivity in many areas. Hence, this significantly bridges the communication and societal status gap.

However, there are constraints due to technological limitations. The main limitations are the implementation of existing software-based convolutional neural network (CNN) models on bulky, heavy and power-intensive platforms such as laptops and personal computers making them unsuitable for daily use. Besides, the hardware implementation of CNN model poses a challenge in the trade-off between the computational demands of CNN and hardware compatibility due to limited computational resources. Therefore, this motivation dedicates an efficient and feasible SL classification hardware prototype for SL classification beyond software simulation and bulky prototype. The implementation on field-programmable gate array (FPGA) as a prototype presents a critical step toward this motivation. FPGA functions as an essential prototype in balancing the design flexibility and performance to validate the architecture of application-specific integrated circuit (ASIC) applications. Final ASIC application will be the ultimate factor for a daily communication aid for the DHH community.

## 1.3 Problem Statement

The implementation of convolutional neural network (CNN) classification model on Field-Programmable Gate Array (FPGA) displays an ideal system with great capabilities to fulfil the demands of real-time applications for sign language (SL) language classification. The main motivation behind the study is addressing the daily challenges faced by deaf or hard of hearing (DHH) community in accessing daily needs, such as social engagement, education and medical healthcare due to communication barriers [3]–[7]. Past research has been focusing on the development of automatic SL recognition system to ease the lives of DHH individuals and promote social inclusivity and accessibility [5], [6], [12], [13].

Most research's CNN model for SL classification achieved high performance at software level but not feasible for hardware implementation. This is due to the complex design of high performing CNN model requires of large memory resource utilization and have high computational cost [2], [8], [12], [14]. This shows the focus only on software model development without the considerations of hardware constraints. Consequently, it displays a gap between software optimized CNN model for SL classification and its implementation on resource-constrained hardware platforms.

In this research, FPGA being a practical solution provides flexibility and scalability addressing the CNN demands with its capabilities of massive parallel processing. While FPGA offer significant benefits to CNN, FPGA also face limitations of scaling to complex CNN models and meets processing needs [8], [15]. Therefore, the CNN architecture requires a systematic hardware-software co-design approach according to the FPGA's resource constraints. The approach ensures the CNN is efficient for hardware implementation.

Furthermore, SL images capture in real-time contains noises from the camera and the environment which necessitate an incorporation of pre-processing filter block within the hardware. Most research apply the filter block at software development level and without explicit indication for hardware implementation for SL classification [5], [8], [13], [16]. In addition, existing co-design filter is not fully optimized and robust for hardware implementation [17], [18].

By addressing these problem statements, this study aims to develop a CNN classification model using hardware-software co-design approach for digit SL classification and implement it on FPGA. Specifically, the research incorporates pre-

processing filtering block which is designed in hardware and co-design the CNN at software CNN development for feasibility verification. Through further research and experimentation, the study aims to develop a feasible SL classification hardware prototype that bridges the gap for the DHH community.

## 1.4 Research Objectives

The research aims to address key challenges in developing sign language (SL) classification using convolutional neural network (CNN) that can be implemented on hardware (i.e., field-programmable gate array (FPGA) as a prototype). To achieve this, the research work is divided into stages that are associated with the research objectives listed as follows :

a) To design and develop a functional digit sign language classification model using convolutional neural network (CNN) such that the architecture can be implemented on hardware.
b) To enhance the performance of digit sign language classification by incorporating an image filtering block in the CNN classification model using hardware-software co-design approach.
c) To evaluate the feasibility of the developed model in (b) to be implemented on hardware for real-time digit sign language classification.

## 1.5 Significance of Study

The study's significance proves an impact upon society, academic knowledge, industry innovation and the expected contributions. The significance addresses issues in deploying convolutional neural network (CNN) model on field-programmable gate array (FPGA) and the application of the system for digit sign language (SL) classification.

By developing an efficient and portable SL classification system, communication is accessible for deaf or hard of hearing (DHH) individuals within the society. This overcomes the exclusivity among DHH community by providing the

society with a communication device in accessing daily needs of communication with DHH community.

Besides, the hardware-software co-design approach for SL classification represents a novel area of study. This study provides further knowledge on the implementation of deep learning models in real-world scenarios by optimizing CNN models on hardware platforms based on the hardware's constraints. At the same time, provides insights and understanding for customization of system that enhance the scalability of the FPGA.

In addition, the study motivates in-depth innovation for embedded systems and assistive technologies. By demonstrating the viability of deploying CNN models on FPGA, this research encourages new industrial applications for more AI-driven solutions, improve existing system and hardware for best performances and wider range of implementations.

Finally, this study presents a feasible FPGA-based CNN classification model prototype for sign language classification. The FPGA prototype serves as a benchmark for future developments that offers a flexible and portable system that can fill the demands various field. By bridging the gap between academic research and practical implementation, the study produces significant contribution in the fields of assistive technology development through FPGA prototyping for application-specific integrated circuit (ASIC) applications.

## 1.6 Research Scope and Limitation

Addressing the research scope is critical in ensuring proper execution of the research within the allocated time frame. With the aid of the scope, the research can be achieved effectively. The research focuses on designing a 7-layered CNN model for sign language (SL) classification that is feasible for hardware implementation. The hardware implemented is the DE2-115 FPGA development board with the Terasic D8M-GPIO camera module only. The SL classification focuses on digit SL which consists only of 10 classes. Therefore, a digit SL dataset is required to train, validate and test the developed CNN. The SL dataset used in this research is obtained from Kaggle platform that contains a total of 5000 images of digit sign language without background noise [19]. The study rely on a noise-free background dataset, which not

fully represent real-world, dynamic environments. Therefore, the dataset sets as one of the limitations of the research. The dataset only focuses on digit sign 0 till 9 whereas each digit consists of 500 images respectively. The research observes the impact of the hardware-software co-design approach of applying pre-processing filter block onto the system. The filters utilized in this research are gaussian, median, sobel and bilateral. The approach begins with designing the filtering block at hardware and then, the same filter block is co-design in the software development. Finally, the research focuses on the feasibility of the overall implementation of convolutional neural network (CNN) with hardware-software co-design approach of filtering block in FPGA prototype and its performance.

## 1.7 Thesis Outline

This thesis consists of five main chapters which are methodically structured to give a flow of understanding of this research from the beginning until the end of the study. The first chapter presents and discusses the research background of the thesis, the problem statement of the research that leads to the research objectives as well as the scope and significance of research. In chapter 2, literature reviews are layout revolving around the research's objectives and problem statement on existing and previous research of convolutional neural network (CNN) based sign language (SL) classification and its hardware implementation.

Chapter 3 explains the methodology and steps taken of this research that aligns with the research objectives. Starting with the preparation of digit SL dataset. Then, begins with the design and development of CNN model architecture and train in MATLAB which reserves as a benchmark result for comparative analysis. The trained CNN model is then implemented in field-programmable gate array (FPGA) via conversion of weights and biased from MATLAB. Real-time testing is conducted upon hardware implementation benchmarking performance. This marks the achievement of the 1$^{st}$ objective that runs a functional CNN model for digit SL classification for hardware implementation. The 2$^{nd}$ objective is achieved by incorporating hardware-software co-design pre-processing filter block to the CNN model for performance enhancement in digit SL classification. The co-design begins at hardware stage and adapts to the filter block design in MATLAB. The same method of evaluating the benchmark results is applied for CNN models with filters for overall evaluation. Finally,

the 3rd objectives achieved through the feasibility of the CNN model with filters in FPGA implementation for real-time digit SL classification. This is further supported by the comparative analysis made with the benchmark result.

Chapter 4 discusses the results of the evaluation from MATLAB to FPGA implementation for real-time digit SL classification. The results dictate the performance and characteristics of the CNN model in comparison to CNN without filter. This highlights and proves the impact and importance of co-design filter block for CNN performance enhancements. The overall performance comparison has been discussed at the end of this chapter.

Finally, chapter 5 draws the conclusion of the research contribution and the needs of future work for this research. This chapter concludes the completion and achievement of research objectives.

# CHAPTER 2
# LITERATURE REVIEW

## 2.1 Introduction

This chapter aims to provide a comprehensive review of the existing body of knowledge and the various approaches on sign language (SL) classification, by focusing on existing and developed convolutional neural network (CNN) model for SL classification and their deployment on different hardware platforms which narrowed down to field-programmable gate array (FPGA). The review begins with the introduction of SL and its challenges. Followed by the classification approaches for SL that narrowed down to the context of digit SL classification. CNN as the subcategory of deep learning models has shown promising results in executing classification tasks which are suitable for SL classifications. Therefore, general architectures of CNN, specific designed models and the application of input pre-processing block as an enhancement development are observed to highlight CNN's capabilities and limitations in this study.

This chapter also addresses the deployment of CNN, its integration on various hardware platforms and the co-design hardware-software method. Then, ultimately narrowed down to FPGA which will be utilized in this study. FPGA has been preferred for real-time deployment of CNN models due to its capabilities of parallel processing that fits the requirement of CNN and outperforms traditional resource-constrained hardware. Key aspects like strengths, weaknesses and specific features of real-time implementation of FPGA are being studied and compared for SL classification to assess its suitability for this study.

By synthesizing the literature, this chapter sets to propose the development of CNN classification model prototype on FPGA for digit SL classification by highlighting the existing gaps in the research and justifying the needs for the study. This review will also establish the theoretical and technical framework for the subsequent chapters of this study.

## 2.2 Digit Sign Language Recognition

Sign language (SL) is a language that uses hand gestures as a means of communication for the deaf or hard of hearing (DHH) community [10]. SL is a method whereas DHH individuals express their thoughts and ideas using hand gestures without verbalizing. There are various SL across the world such as American SL [3], [12], [20]–[29], Bengali SL [20], [30], Korean SL [1], [31], [32], Arabic SL [4], [12], [33]–[41], Norwegian SL [5], Bangla SL [13], [42]–[44] and Indonesian SL [16], [45], [46]. SL has been known to foster cultural identity for DHH community in various countries by promoting better social integration, mental health and educational outcomes. However, DHH community is still at disadvantages and isolated from society. This is due to the communication barrier between DHH community with the rest of the world [3]–[6]. The lack of SL proficiency and availability of SL interpreters poses more challenges [5], [6], [20], [40], [43].

In addressing the communication gap in the development of automated SL classification system, designed SL classification system studies focused on recognizing both letters of the alphabet and numerical hand sign. The variability of hand signs approach faces significant challenges due to the complexity of SL gestures styles and variations of hand positioning and angles [1], [4], [42], [47], [48]. Various machine learning models like Support Vector Machine (SVM) classifier [7], K-Nearest Neighbors (KNN) [5], ensemble techniques of combining several learning algorithms [49] and convolutional neural network (CNN) [36] has been studied in addressing the challenges These approaches have shown a good accuracy performance of SL classification. Table 2.1 shows studies on different focused SL data across different SL.

Within the studies of SL classification, digit SL emerges as a specific focus area. Digit SL is one subset of sign language that emerges as a manageable and practical focus area. Digit SL dataset consists of 10 classes (0-9) that offers less complexity compared to alphabets of 27 letters, making them suitable for benchmarking of new develop model due simplicity and achievable high performance [28], [31], [36], [37], [42], [43]. Digit SL is recognized universally and are commonly implemented in real-world applications such as interpreting prices, and addresses [21], [28], [31], [48].

Table 2.1
Selected Past Literature to the Architecture of CNN Models for SL Classification

| Study | Sign Language (SL) | Machine Learning | SL Dataset | Results |
|---|---|---|---|---|
| Svendsen et al. 2023 [5] | Norwegian SL | KNN, CNN & SVM | Letters | KNN : 99.90%<br>CNN : 99.90%<br>SVM : 99.90% |
| Alharthi et al. 2023 [12] | Arabic SL | CNN | Letters | 63.23% |
| Jain et al. 2021 [22] | American SL | SVM, CNN (Single & Double layer) | Letters | SVM : 81.49%<br>CNN : 98.58% |
| Adeyanju et al. 2022 [29] | American SL | KNN | Letters | 99.00% |
| Tharwat et al. 2021 [35] | Arabic SL | KNN | Letters | 99.7% |
| Mahmoud et al. 2024 [39] | Arabic SL | Optimized hybrid CNN and ML | Letters | 98.90% |
| Shams et al. 2024 [13] | Bangla SL | Static-weight averaging Ensemble | Letters | 95.13% |
| Nihal et al. 2021 [44] | Bangla SL | Zero Short Learning | Letters | 91.75% |
| Sari 2023 [16] | Indonesian SL | Mnasnet1_0 | Letters | 85.75% |
| Rebrastya et al. 2025 [45] | Indonesian SL | CNN | Letters | 87.1% |
| Rahim et al. 2022 [30] | Bengali SL | Ensemble model (CNN+RF+SVM) | Digit | 99.50% |
| Miah et al. 2022 [20] | Bengali SL | CNN | Letters & Digit | 99.99% |
| Hariharan et al. 2025 [28] | American SL | Modified deep residual CNN | Letters & Digit | 95.00% |
| Shin et al. 2024 [31] | Korean SL | Resnet101 | Letters & Digit | Letters : 92.04%<br>Digit : 93.88% |
| Ismail et al. 2021 [36] | Arabic SL | DenseNet121, VGG16 | Letters & Digit | 99.99% |
| Ahmadi et al. 2024 [37] | Arabic SL | CNN | Letters & Digit | 97.02% |
| Das et al. 2023 [42] | Bangla SL | Hybrid (VGG16 + RF) | Letters & Digit | Letters : 97.33%<br>Digit : 91.67% |
| Podder et al. 2022 [43] | Bangla SL | ResNet18 | Letters & Digit | 99.99% |
| Sharma et al. 2021 [21] | Indian SL | Gesture-CNN | Letters & Words | 99.96% |
| Bencherif et al. 2021 [33] | Arabic SL | OpenPose Network Architecture | Words, Letters & Digit | 89.62% |
| Shin et al. 2023 [1] | Korean SL | Transformer-based deep neural network + CNN | Words | 93.30% |
| Balaha et al. 2023 [4] | Arabic SL | CNN + RNN | Words | 98.00% |
| Castro et al. 2023 [3] | Brazilian SL | CNN | Words | 91.00% |
| Shin et al. 2023 [32] | Korean SL | Graph Convolutional Network (GCN) | Words | 100% |
| Abdul et al. 2021 [34] | Arabic SL | CNN Inception + BiLSTM model | Words | 85.60% |
| Al-Onazi et al. 2023 [38] | Arabic SL | Deer Hunting Optimization | Words | 92.88% |
| Noor et al. 2024 [40] | Arabic SL | CNN | Words | 94.40% |
| Saputra et al. 2024 [46] | Indonesian SL | YOLOv5 | Words | 76.42% |

Furthermore, digit SL gestures are easily identified and recognizable compared to dynamic gestures of word signs due to its static pose and unnecessary needs of continuous movement [21]. This produces higher accuracy for SL classification especially in benchmarking of a new develop SL classification system.

Focusing on digit SL gives an effective foundation for the development of SL classification system. The convenience that digit SL offers makes them an excellent starting point of developing scalable and efficient classification models. This also offers current systems a learning opportunity to handle complex SL datasets in the future. By narrowing down the focus to only digits SL, this study can achieve significant progress in understanding the theoretical and practical application of the SL recognition technology.

In classifying digit SL effectively, a suitable machine learning model is required and among the common model used is CNN . CNN is an excellent approach for SL classification due to its capability to extract spatial and hierarchical features from raw input image data. Compared to other deep learning models, CNN is able to learn patterns directly from the extracted raw data. This capability can enhance CNN architecture to overcome complex structures and various hand gestures found in SL datasets [13].

There is much evidence that proves the effectiveness of CNN model in SL classification based on previous research. For example, a study conducted on Bengali SL alphabets achieved 99.99% and 94% for training and testing accuracy respectively, resulting from the developed CNN model named BenSignNet with 20 number of layers [20]. Other researchers developed a 12-layered CNN model to classify 2 different SL of Indian and American SL datasets that yields high recognition accuracies of 99.96% and 100% respectively [21]. These examples of past studies prove the adaptability of CNN on various SL datasets, further show its viability of SL classification approaches.

By focusing on digit SL datasets, CNN models are able to provide firmer foundation for benchmark scalability and adaptability of developed SL classification system. This enables the system for further modifications of SL classification system that delve into more complex SL gestures and patterns. This capability of CNN proves as a pivotal technology for advance SL classification in future research on this field of studies.

## 2.3 Convolutional Neural Network (CNN) for Sign Language Classification

Convolutional neural network (CNN) represents a complex class of deep learning models for processing, analyzing and interpreting data, most commonly visual images [50]–[55]. Fundamentally, CNN functions automatically and adapts to learn complex features of data which enable CNN to detect complicated patterns and structures in raw input data [56]. This capability of CNN is done through a series of layers that includes convolution and pooling operations. Convolution operation helps to extract local features such as edges and textures. Meanwhile, pooling operation reduces the spatial dimensions of the data which preserves the primary and initial data information. The extracted features are compiled and accumulated to form a more abstract representation of the image which will be used for desired CNN-based task such as classification [8], [10], [11], [15], [50], segmentation [20], [57], [58], recognition [47], [59], [60], and feature extraction [61]–[64].

CNN have proven successful demonstrations across the field of studies by processing and analyzing high resolution image data efficiently. In medical healthcare sector, CNN are implemented for medical image analysis such as, classification of histopathological images that detects cancer, coronavirus diseases (COVID-19) detection and X-ray images classification [65]–[69]. In agriculture field, CNN are deployed for crop disease classification with enhanced agricultural decision-making system for farming precision to improve the quality of harvest products and image-based species classification that classify wide variety of harvest [54], [70]–[72]. Meanwhile in autonomous systems, CNN are installed on self-driving vehicles for real-time object detection which contributes to safe drive navigation on the road [73]–[75]. The capacity and efficiency of CNN's end-to-end learning makes it the preferred instruments to be implemented in wide range of industries from autonomous systems and surveillance to healthcare and agriculture [8], [71], [76], [77].

CNN classification models have excelled at learning hierarchical representations of input data compared to traditional machine learning models that require large scale and lengthy engineering features [50], [75]. With structured designs of layers such as convolutional layers, pooling layers and fully connected layers, CNN is able to learn spatial hierarchies of features that help to detect complex patterns in visual data. Thus, makes CNN ideal for image classification tasks. This review aims to provide thorough overview of CNN classification models by discussing various CNN

models and its architectures on image classification. With further study of CNN implementation on sign language (SL), this review seeks a deep understanding of its adaptability for SL classification.

### 2.3.1 CNN Architecture

Convolutional neural network (CNN) is typically designed to imitate the human visual processing system, that enables the network to learn and extract features from input data, particularly images. A standard CNN consists of multiple layers such as input layer, convolutional layers, pooling layers, and fully connected layers that help to perform classification task [53]. A sample lightweight CNN architecture taken as an example used for age estimation and gender classification is shown below in Figure 2.1 [54]. The convolutional layers are responsible for extracts raw spatial hierarchies in the data, while pooling layers help reduce computational complexity and prevent overfitting [74], [78]–[81]. The fully connected layers make final decisions and conduct the classification process.

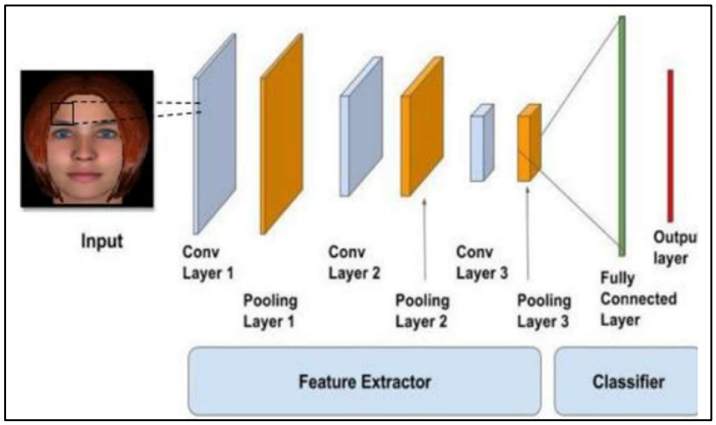



Figure 2.1 Lightweight CNN Architecture Age Estimation And Gender Classification [54]

With the referred state-of-the-art architecture framework, the design of CNN has evolved and customed based on different types of tasks and field applications. Popular CNN models such as VGG-16, ResNet, and AlexNet are frequently utilized by previous research for conventional image classification or as a benchmark model due to their ability to learn hierarchical features [66]. For more intricate models, researchers used ResNet152 and Wide Residual Network (WRN) that utilize deep layers and residual connections to overcome the issues like losses of gradient [82]. For example, the Inception-ResNet-v2 module is a combination of Inception modules and residual

connections which designed to enhance the task performance to recognize complex image like weed species identification for an effective site-specific weed management [83]. Additionally, lightweight CNN model like MobileNetV2 and NASNet-Mobile are designed for mobile and embedded devices that provide effective trade-off for both the computational cost and accuracy [57]. Lightweight CNN architecture has lesser number of layers in comparison to other models as it focused on reducing latency and model size while maintaining accuracy. Other models are developed with specialized architecture that designed from the ground up for a specific task like YOLOv3 and Faster R-CNN which made specifically for object detection that enables real-time localization [73]. One example is eCNN which used to specifically design for embedded systems with constraints of power consumption and computational cost [8].

Each of these models is applied across various fields of study, including computer science and engineering, agriculture, medicine, transportation engineering, robotics, cybersecurity, remote sensing, and more. The flexibility and adaptability of CNN architectures design allow CNN models to be leveraged across various domains, improving their performance and enhancing their capabilities. Table 2.2 listed the applications of CNN models in various fields along with the previous research reference.

Table 2.2
Selected Past Literature to the Architecture of CNN Classification Models

| Study | Title | CNN Model | Field of Study |
|---|---|---|---|
| Wang, CC et al. 2021 [8] | Real-Time Block-Based Embedded CNN for Gesture Classification on an FPGA | Embedded convolutional neural network (eCNN), VGG | Computer science |
| Dutta, TK et al. 2024 [66] | ARM-Net: Attention-guided residual multiscale CNN for multiclass brain tumor classification using MR images | AlexNet, VGG19, GoogLeNet,VGG-16, VGG-19, ResNet-50, Xception, Inception V3, EfficientNet-B0, SqueezeNet, MobileNet | Medical image analysis |
| Satyanarayana, GSR et al. 2022 [73] | Vehicle detection and classification with spatio-temporal information obtained from CNN | Faster R-CNN, ZF-NET, YOLOv3 | Intelligent transportation systems and computer vision |
| Chen, X et al. 2021 [82] | Cyclic CNN: Image Classification with Multiscale and Multilocation Contexts | ResNet, VGGNet, Wide Residual Network (WRN) | Computer vision and deep learning |
| Mesías-Ruiz G.A et al. 2024 [83] | Weed species classification with UAV imagery and standard CNN models: Assessing the frontiers of training and inference phases | VGG16, ResNet152, Inception-ResNet-v2 | Agricultural science |

### 2.3.2 Impact of Pre-processing for CNN

Convolutional neural network (CNN) models have proved to performed well in various fields due to its efficient ability to learn complex features of images. In addition to improves the model's performance and reduces the computational cost of a CNN, input pre-processing block is introduced to the system. A pre-processing block helps to improve image quality and reduce noise to ease the models in enhancing feature extraction process. There are various techniques in pre-processing and the common methods used is data augmentation such as scaling and normalization [84]–[86]. Scaling is resizing the images based on the adaptation of the CNN model that gives a consistent dimension that involves various sizes of samples such as past research on different images' dimensions of gastrointestinal tract diseases [84]. After scaling, images are normalized and this process is called normalization. Normalization process is common in many pre-processing whereby it involves the pixel values of an image to be normalized to a standard range of [0,1] that maps the intensity of image pixels [86]. This ensures a uniform input for the CNN and improves convergence during the model's training. Besides, the other common data augmentation process is rotating images or flipping the images which helps to prevent overfitting and improve generalization for CNN during training [12], [13].

The second most common pre-processing method is noise reduction filter technique. This technique applies various filters such as gaussian, median or bilateral filtering to reduce noise of images and smoothen images by preserving edges and important details [85], [87]–[89]. The denoise of images is particularly crucial in many fields such as a medical diagnosis where image clarity is needed for specific attention. A detailed discussion of noise reduction filters' function and advantages are elaborated in section 2.3.3.

Another method of pre-processing is called contrast enhancement. Contrast enhancement is almost similar to normalization except it uses techniques involving histograms [29], [50], [84]. The use of histograms is useful for images that affected by illumination where the image is equalized to improve the image contrast. Despite the differences of pre-processing techniques used in different studies, some studies choose to utilize more than 1 techniques to further improve the performance of the CNN [84], [85], [88]. The impact of the applied pre-processing techniques improves the performance of CNN models respectively and is shown in Table 2.3. Thus, shows the

importance of pre-processing techniques in achieving enhanced performing CNN model.

Table 2.3
The Implementation of Pre-Processing in CNN Models

| Study | Title | CNN Model | Pre-processing | Impact of pre-processing |
|---|---|---|---|---|
| Zhou et al. 2022 [88] | HCCANet: Histopathological image grading of colorectal cancer using CNN based on multichannel fusion attention mechanism | HCCANET | Rotation, cropping, scaling, noise reduction using filters (mean, median, gaussian, and bilateral) | Increased training data, noise reduction |
| Ahamed et al. 2024 [84] | Detection of various gastrointestinal tract diseases through a deep learning method with ensemble ELM and explainable AI | Ensemble Extreme Learning Machine (EELM) | Histogram, gaussian filter, image erosion, sharpening | Enhanced contrast, noise reduction, edge clarity, sharpen image |
| Islam et al. 2022 [85] | Multimodal hybrid deep learning approach to detect tomato leaf disease using attention based dilated convolution feature extractor with logistic regression classification | Attention-based dilated CNN | Colour conversion, bilateral filter, Normalization, Otsu segmentation, synthetic image generation. | Noise removal, feature clarity, balanced dataset, |
| Balasubramanian et al. 2023 [90] | APESTNet with mask R-CNN for liver tumor segmentation and classification | APESTNet with Mask R-CNN | Histogram equalization and median filtering | Boost contrast, noise reduction |
| Le et al. 2021 [87] | IoT enabled depthwise separable CNN with deep SVM for COVID-19 diagnosis and classification | Depthwise separable CNN (DWS-CNN) | Gaussian filter | Noise elimination, improved image quality |

In sum, pre-processing method is a critical foundation for optimizing CNN performance in many applications. Among all the methods, noise reduction filter consistently and commonly applied due to its denoising ability and enhancing image quality. As shown in Table 2.3, research in various domains integrate filtering techniques such as gaussian, median and bilateral filters in eliminating noise and enhance the image. Another adoption of this method is combining it with other pre-processing methods like data augmentation and contrast enhancement that sets a fundamental step in preparing the input data for the training of CNN.

### 2.3.3 Pre-processing Noise Reduction Filter for CNN

Noise reduction filter has been widely used across diverse data types including medical images [89], [91]–[93], hyperspectral remote sensing data [94], and

microscopic images [85]. The wide application proves its robustness and generalized utility specifically on its contribution on the performance of convolutional neural network (CNN) models. The advantages of using filters across various domains is the effectiveness of noise suppression while preserving features. Filters like gaussian, median and bilateral are common filters used to reduce noise and preserve edges of images. Research that used bilateral filtering proves that images studied are smoothen while the edge sharpness are maintain [85]. Bilateral filters use a weighted mean of intensity data based on the surrounding pixels to adjust the pixels brightness of the focused subject. This is crucial especially for medical image analysis like malaria parasite detection [92] and melanoma detection [93] to ease the identification of disease on a microscopic level. In addition, the bilateral filter also is beneficial for microscoping imaging in the agriculture field like tomato leaf disease detection [85]. Similarly, median filter is also utilized in medical image analysis for superior noise reduction and edge preservation. Median filtering is a non-linear filter that substitutes noisy pixels with a median value based on the neighbouring pixels. Example studies using median filtering are applied on brain tumor [91] and lung x-ray images [89]. Another research uses gaussian filter as a linear smoothing filter that applies gaussian average weight to neighbourhood pixels in the medical field which smoothen the image data and effectively reduces noise [87]. Gaussian filter has effectively reduced noise while preserve important details for colorectal cancer histopathology using neuroimaging, hence improve the effectiveness of diagnostic abilities [88].

Another key advantage of using noise reduction filter is due to its simplicity and computational efficiency. Applying a filter is simpler and computationally light compared to complex techniques like algorithms or deep learning-based denoising. In the studies of tomato leaf disease detection, the researchers described the application of bilateral filter combined with the Otsu method of segmentation as rather fast and simple approach for noise removal as both methods compliments one another in removing noises efficiently [85]. In conjunction with the combination techniques, this brings to the advantage of filter integration with other techniques with ease. Filters are easily combined with other pre-processing methods that further improve the data quality as seen in the works of ensemble learning and multimodal approaches [84].

In sum, noise reduction filter technique is a reliable technique as it reduces noise while preserve important features of images. At the same time, it contributes an efficient computation by reducing the computational cost to models like CNN, widely applicable

in various fields and integrates well with other methods. Thus, the application of filter is reliable choice for pre-processing study especially in deep learning that involves image analysis.

### 2.3.4 Sign Language Classification CNN Models

With the aid of convolutional neural network (CNN), sign language (SL) can be interpreted for deaf or hard hearing (DHH) community to bridge the communication gap. CNN models have been widely used to extract features and classify SL patterns and display the meaning of every SL sign and gesture [12], [13]. CNN classification models developed by previous research that has been refined for SL classification are tabulated in Table 2.4. Despite the CNNs' capabilities to decode SL into text or speech, there are few challenges to be addressed such as gesture variability and real-time performance [3], [8], [11], [12], [21], [34]. Therefore, research applied various techniques such as transfer learning, and multi-modal approaches to bridge the communication gap.

Table 2.4
CNN Model Approach on Sign Language

| Study | Sign Language | CNN Model | Pre-processing | Accuracy |
|---|---|---|---|---|
| Wang C et al. 2021 [8] | American SL | VGG-based RGB-D CNN model & proposed RGB-D eCNN model. | Skeleton tracking, Normalization | 99.79% |
| Svendsen B et al. 2023 [5] | Norwegian SL | SVM, KNN, and CNN models | Data augmentation, segmentation | 99.9% |
| Sahoo J et al. 2022 [14] | American SL | AlexNet & VGG-16. | Scaling, segmentation, depth thresholding | 98.14% |
| Alharthi N et al. 2023 [12] | Arabic SL | CNN-1, CNN-2, & CNN-3 | Segmentation, Normalization | 95.56% |
| Shams K et al. 2024 [13] | Bangla SL | **Model-1:** built on a sequence of Conv2D, Batch Normalization, MaxPool, Dropout, Flatten, and Dense layers. **Model-2 & Model-3:** use the same architecture with changes in the batch sizes during training | Canny edge detection | 95.13% |
| Sari I 2023 [16] | Indonesian SL | Inception_v3, Shufflenet_v2x1_0, Densenet161, Densenet201, Wide resnet50_2, Mobilenet_v2, & Mnasnet1_0. | Scaling, Data augmentation | 85.75% |

However, the high-performance CNN model in past research is typically trained based on software development level where it depicts an ideal environment for

evaluation without the consideration of hardware constraints [48], [95]. The hardware implementation for CNN must consider factors such as memory size and computational complexity as limitations. CNN models with complex architecture may consists of parameters that exceeds available resources within a hardware. Therefore, a hardware-software co-design approach is required in order for CNN to be implemented efficiently and fully optimized the hardware constraints.

### 2.3.5 Hardware-Software Co-design of CNN for Hardware Implementation for Digit Sign Language Classification

The previous sections have established several key observations that points to a central challenge in developing convolutional neural network (CNN) sign language (SL) classification system for hardware implementation. In earlier section, CNN's complex architecture is highlighted which varies significantly based on different parameters that demands large computational resources and memory. Moreover, pre-processing methods of applying noise reduction filter are also highlighted due to its capabilities of enhancing input image and improve CNN performance by preserving important features of image data. The addition of the filter is accounted for the increase of computational cost in developing the SL classification system. Furthermore, a critical observation is revealed in existing SL classification research where most system achieve high performance consistently at software development level and hardware constraints are not explicitly considered or mentioned for the development of CNN. In addition, the pre-processing method are implemented exclusively in software only which further shows the neglect the consideration of available hardware resources for hardware implementation of CNN.

The observation displays a fundamental gap between software-oriented development and the practicality of hardware implementation. This is a common approach where CNN is first developed for maximum performance at software level and later adapted to the hardware constraints. This approach often is inefficient and requires a complete redesign. This is unsuitable for resource-constrained hardware implementation where the available resources are limited based on the selected hardware. High performing CNN with deep layers and large parameter counts are not well suited for the limited hardware implementation.

Therefore, a hardware-software co-design approach is introduced. This approach is a development of CNN that consider the constraints and capabilities of hardware [18]. For a CNN-based SL classification system that targets its implementation on a resource-constrained platform, this method involves several considerations. The first consideration is understanding the hardware available memory and processing elements. This ensure the CNN layers, filter sizes and parameters are optimized fully within the hardware [8], [15]. The next consideration involves the quantization strategies applied in hardware and replicate the same strategy to software development. The quantization strategy lowers the memory footprint in hardware and reduces computational complexity of CNN [48], [96]. With the hardware constraints consideration, CNN is optimized with understanding of the hardware implementation details such as parallelization patterns that covers potential loss in classification [11]. Looking at the incorporation of pre-processing filter, the co-design approach enables a well pipeline execution in the hardware where the input data streams to the filter then send directly into the CNN and allow sharing on-chip memory buffers between the filter and CNN [17], [18]. The hardware-software co-design approach provides a framework in developing a SL classification system where both CNN architecture and incorporate pre-processing filter are optimized for efficient hardware implementation on a resource-constrained platform.

## 2.4 Hardware Implementation of CNN Classification

The hardware implementation for convolutional neural network (CNN) deployment varies based on CNN demands and field application. The choice of hardware represents in 3 categories of fundamental focus which are performance acceleration, energy-efficient embedded execution and general-purpose programmability.

For intensive computation tasks like training complex model or running high-throughput data inference, specialized hardware accelerators are required. High-Graphics processing units (GPUs) like NVIDIA GeForce RTX and GTX (30-series and 20-series) GPUs and Tesla K40, capable of attaining high computational power and memory capacity useful for extensive training of CNN [57], [81], [83], [97]. These GPUs with big of memory, execute convolutional operations efficiently in CNN.

However, CNN deployment in real-time demands a hardware that balances power efficiency and its physical sizes. Therefore, embedded systems is a solution for real-time deployment. Platforms such as NVIDIA Jetson series (TX1, TX2, and Nano), which are most preferred by consumers, integrate moderate-performance accelerators with Advanced Reduced Instruction Set Computer (RISC) Machine (ARM) based central processing units (CPU), which is suitable for a low to mid performance CNN models execution [69]. Similarly, Raspberry Pi models offer low-cost solutions for straightforward task-oriented models [98]. With refiner system like embedding GPUs with a microcontroller (eg. STM32F407) capable of providing a flexible edge computing and internet-of-things (IoT) applications as additional features to the CNN application [99].

In contrast, FPGA is a hardware platform that balances performance and efficiency through customization. FPGA-based systems, such as the ZCU 102 SoC evaluation board or the ZYNQ Ultrascale ZCU106 FPGA, are a hybrid hardware where they are reconfigurable and often deployed in embedded system. This allows optimization of CNN operations into custom digital circuits [8], [10], [15]. Thus, resulting in a performance efficiency and design flexibility.

In summary, the selection of hardware implementation for CNN depends on the strategic needs of computational power and energy efficiency. Hardware platforms offer possibilities to accelerate the performance of CNN across all domains based on the constraints that compliments the need of CNN. Table 2.5 listed the selected past literature of hardware implementation of CNN classification models.

Table 2.5
Selected past Literature of Hardware Implementation of CNN Classification Models

| **Study** | **Hardware Platform** | **Hardware design fundamental focused** |
|---|---|---|
| Lu, Y. et al. 2021 [81] | Tesla K40, GTX 1080 Ti GPUs | High performance acceleration |
| H. Gao. et al. 2018 [74] | NVIDIA GeForce GTX Titan X and NVIDIA Jetson TX1 | Comparative analysis (acceleration vs embedded) |
| Satheeshkumar K.G. et al. 2024 [69] | NVIDIA Jetson Nano | Energy efficient embedded execution |
| Cho, SI 2020 [98] | Raspberry Pi 4 | Low cost embedded execution |
| Rawal V. et al. 2023 [10] | ZYNQ Ultrascale ZCU106 FPGA | Customizable hardware acceleration |
| Sabor N. et al. 2022 [99] | STM32F407 microcontroller | Low power embedded control |
| Li, QL et al. 2024 [100] | Intel Xeon and NVIDIA GeForce RTX 3080 | High performance acceleration |

### 2.4.1 Hardware Implementation of CNN for Sign Language Classification

A controller is an essential component responsible in managing and coordinating operations in any system. In every system, controller works as the central unit that executes, commands, process data and conducts interfacing with other hardware parts. In the context of convolutional neural network (CNN), controllers play a crucial role for the execution of CNN models, especially in resource-constrained environments. Therefore, the advantages and disadvantages of controllers must be considered to ensure CNN's deployment and performance efficiencies.

One of the advantages of controllers in CNN deployment is the high computational efficiency and short training time of CNN. Controllers are capable of executing multiple operations simultaneously which speed up the training time and inference processes of CNN models [81]. This allows efficient deployment of CNN complex models, thus making controllers suitable for real-time applications. Moreover, controllers are capable of handling computational demands associated with training complex CNN models. By relieving computationally intensive tasks to assigned processing units, controllers can manage large datasets and operations efficiently with low latency and high throughput [51], [101]. Furthermore, in some cases, controllers offer a low-cost solution with lightweight hardware where high computational power is not necessary such as embedded systems, Internet-of-Things (IoT) devices and low-cost applications [4], [14], [73], [102].

On the other hand, the disadvantage in using controllers is the scalability of the computational resources. As the architecture of CNN grows in complexity with demands of advanced applications, the processing power and memory of controllers becomes insufficient and leads to performance obstruction. With improper optimization, overfitting occurs during training [3], [6], [12], [14], [21], [34]. In achieving the optimal performances of CNN, specialized or high-performance hardware is required, which may not often be available in a standard controller build. Additionally, the need of additional components to support complex operations may increase the hardware overhead [73], [81]. Without the required hardware, controllers' capabilities will not meet the demands of advanced and complex CNN models. Finally, controllers have higher power consumption for computationally intensive tasks which raise concerns for CNN applications that require efficient energy for a mobile and portable system such as mobile or battery powered devices [6], [69], [74], [83], [102].

While controllers provide significant advantages of deploying CNN models, there are few disadvantages that need to be considered carefully. The success of deploying CNN models on controllers depends on the application, needs of resources and performance. As technology advances, standard controllers underperform and are unable to deliver an optimal performance and efficiency for hardware implementation of CNN in controllers. With the limitations of controllers, Field-Programmable Gate Array (FPGA) becomes an alternative that suits the complex demands of CNN. FPGA is a manufactured integrated circuit that can be configured by any user or consumer. This allows FPGA to be reprogrammed to perform custom implementation of digital circuits and hardware accelerators, giving flexibility in design and its functionality [8], [10], [96]. Therefore, FPGA is adaptable for a wide range of applications such as digital signal processing, wireless communication, embedded systems, networking and control systems. FPGA is most favored to deploy convolutional neural network (CNN) model because of its flexibility in reconfigurations [103].

FPGA's main capabilities which ideal for CNN deployment are its massive parallel processing design whereby multiple operations can be executed simultaneously and speed up the computation process [8], [10], [11]. The parallel processing capabilities are effective for CNN's convolution operation that applies on the input images which allows execution of thousands multiply-accumulate (MAC) operations in parallel across the hardware circuits. Additionally, FPGAs have low power consumption compared to conventional processing units, thus making it suitable for edge computing applications [8], [96], [104]. The low power consumption in FPGA stems from the custom design of data paths on necessary computations only for a given CNN model which eliminates the overhead instructions. Furthermore, CNN model inference depends on the high-throughput data channels of FPGAs, which can effectively manage massive data volumes [11]. This is due to FPGA's configuration of custom memory hierarchy in memory interfaces and on-chip buffers of block random access memory (BRAM) for minimal data transfer bottleneck and continuous data flow to the processing engines.

To deploy CNN classification models on a FPGA, the CNN must fulfill the hardware requirements. This involves the optimization of the CNN's architecture design to attain hardware constraints such as processing power, memory sizes and logic resources availability. The deployment process also involves the conversion of the weights and biased from the trained CNN model into a 16-bit fixed point format

implemented in the FPGA. This includes designing the hardware accelerators for CNN layers like convolution layers, pooling layers and fully connected layers. Additionally, CNN models typically involve a large dataset to gain more accurate end results. Thus, a high memory bandwidth is required, and FPGA is able to fill those necessities by customizing its memory hierarchy [48], [103].

The integration of CNN classification models with FPGA serves several purposes. A key justification for using FPGA in CNN deployment is the ability to provide real-time inference with low latency, which is essential for real-world applications like medical imaging, robotics, and driverless cars [9], [10]. FPGA integration's adaptability makes it possible to optimize for any application requirements, including higher performance in high-throughput activities or reduced power consumption in embedded systems [9], [96]. In comparison to the general-purpose controllers, FPGA offers a hardware-level approach to acceleration due to its accessible configuration as a prototype for a manufactured integrated circuit by any user. At the same time, this fulfils the realization for application-specific integrated circuit (ASIC) application.

### 2.4.2 Real-Time Hardware Implementation of CNN for Sign Language Classification

With Field-programmable gate array (FPGA) capabilities demonstrated across various applications, this establishes its viability for classification task. This section reviews the general implementation of convolutional neural network (CNN) classification in FPGA and further specified for sign language (SL) classification. FPGA-based CNN classification models have been developed across various fields. For instance, the FPGA-based CloudSatNet-1 model for satellite on-board cloud coverage classification displays FPGA's suitability in extreme environments like outerspace by using radiation-hardened board while achieve high accuracy (94.4%) with low power consumption of 2.5W [96]. Besides, another FPGA-based 1D-CNN model developed for ECG arrhythmia classification is able to conduct heart disease detection efficiently and accurately by leveraging FPGA's parallelism which demonstrates the use of CNN in medical diagnostics. This is a notable implementation of FPGA's low power consumption (0.58W to 0.628W) and high accuracy detection (97.34% to 99.3%) in

rapid decision-making ECG classification inference engine for mobile healthcare applications [10], [105].

Beyond aerospace and healthcare, FPGA is efficient in resource constrained settings. A low-power CNN-based environmental sound classification processor on Kintex-7 FPGA demonstrates this by operating in high accuracy (84.5%) while consuming 0.313W, making it suitable for IoT edge devices with constrained computational resource and power budgets [104]. Furthermore, Chaos LiDAR-based RGB-D was designed by previous researchers for face classification systems. The model demonstrates integration of CNN for face recognition tasks using depth sensors on FPGA achieves high accuracy (99.77%) with minimal latency (11.11ms) and low power consumption (2.195W) [11]. Therefore, providing security and surveillance systems in depth. Moreover, real-time gesture classification is another application of FPGA based CNN classification model, where block-based embedded CNN are deployed that shows hardware optimization yields accurate gesture recognition (99.79%) in interactive systems [8].

Various types of FPGAs implemented are optimized based on the desired performance of the field research. For example, the Kintex-7 FPGA and Xilinx Artix-7 FPGA are known for the trade-off between power consumption and performance [10], [96], [104]. Meanwhile, the ZYNQ Ultrascale ZCU106 FPGA and Xilinx Zynq-7000 SoC ZC706, with embedded advanced reduced instruction set computer (RISC) machine (ARM) processors and programmable logic are designed for more complex applications due to its flexibility of hardware-software co-design and efficient hardware capacity [8], [10], [105]. The hardware-software co-design in FPGA for CNN is necessary for real-time deployment for CNN. The main reason is to leverage FPGA's acceleration for CNN's intensive computation task and software flexibility for data handling. Although FPGA has a massive parallel processing, the resources are still limited [106]. Co-designing the CNN balances computation and memory usage by caching intermediate results [48]. Besides, co-designing reduces lengthy design cycles by enabling easier updates and model management.

In sum, FPGA-based CNN systems are widely implemented across number of industries, including healthcare, environmental monitoring, gesture recognition, face recognition, and wireless communication. The deployment of CNN into FPGA produces effective real-time classification systems and have them applied in areas such as mobile healthcare, autonomous systems, and Internet of Things (IoT) devices.

Moreover, the existing system produces different results based on the respective applications. Table 2.6 shows the past literature of FPGA based CNN classification models with its classification type and performance.

Table 2.6
Past Literature of FPGA based CNN Classification Models

| Study | Type of FPGA | CNN Model | Classification Type | Performance |
|---|---|---|---|---|
| Wang C et al. 2021 [8] | ZCU 102 SoC evaluation board | RGB-D Block-based embedded convolutional neural network (eCNN) | Gesture Classification | **Accuracy** : 99.79% **Power Consumption** : 2.664W |
| Pitonak R et a. 2022 [96] | Zynq-7020 FPGA board, Zturn development board & Xilinx Kintex Ultrascale XQRKU060 board | CDNetV1, CDNetV2, & CD-FM3SFs | Cloud Coverage Classification | **Accuracy** : 94.4% **Average Power Consumption** : 2.5W |
| Peng L et al. 2023 [104] | Environmental sound classification processor on a Kintex-7 FPGA | Big-small CNN-based reconfigurable ESC processor | Environmental sound classification | **Accuracy** : 84.5% **Power Consumption** : 0.313W |
| Rawal V et al. 2023 [10] | Xilinx Artix-7 FPGA & ZYNQ Ultrascale ZCU106 FPGA | SqueezeNet, CNN-KNN, CNN-LSTM, CNN-RNN & 164 different 1D-CNN | ECG arrhythmia classification | **Accuracy** : 97.34% **Power Consumption** : 0.628W |
| Chiu C et al. 2022 [11] | ZCU102 SoC | RGB-D eCNN | Face classification | **Accuracy** : 99.77% **Power Consumption** : 2.195 W |
| Zhang C et al. 2023 [105] | Xilinx Zynq-7000 SoC ZC706 evaluation board kit | Two 1-D CNN | ECG classification | **Accuracy** : 99.3% **Power Consumption** : 0.58W |

The general application of FPGA sets a foundation for researchers to focused on FPGA-based CNN implementation for sign language classification in particular. Since FPGA proves as a viable solution to deploy CNN model in real-time, it is essential to conduct a review on previous works of different types of FPGA and their performance in SL classifications execution. Table 2.7 shows the metrics that were focused on to evaluate the performance of FPGA across different studies. It is crucial to evaluate a range of evaluation metrics to determine the effectiveness of a FPGA-based CNN model for SL classification. One of the metrics is the clock speed (MHz). The clock speed shows the computations speed perform by the FPGA and proves its fast processing for real-time applications. The highest clock speed in past study was performed in Arria 10 GX1150 with 240MHz [8]. Higher clock spend enables the FPGA to handle its computation process efficiently by reducing the time taken for inference, thus improving the overall system's performance. Despite the high clock speed, it may cost a high computational power cost to feed the required processing speed. Therefore, the next metrics to look at is the power (W) consumption of the FPGA.

Power consumption is a vital factor for a hardware to operate by constraining its energy consumption while delivering high performance. The lowest power consumption ever used for SL classification previously will be the PYNQ-Z2 ZC7Z020 with 1.8W [103]. Additionally, the performance of FPGA is evaluated based on the giga operations per second (GOPS). GOPS indicates the FPGA's total number of operations per second, and it reflects the computational cost of the FPGA and its ability to handle complex CNN model operations efficiently. Moreover, the balance of computational power and resource used within the FPGA can be evaluated further based on the power efficiency (GOPS/W). The power efficiency shows the results of complete operations per unit power consumed. With these metrics, ZCU 102 SoC board gives a promising performance while consume less power for SL classification with 2.664W power consumption, 512 GOPS and 192.19 GOPS/W [8].

Furthermore, the number of digital signal processing (DSP) blocks is another factor in contributing to a high performance of FPGA for CNN deployment. DSP blocks are specialized hardware resources within the FPGA that plays a significant role for efficient computation. With more DSP blocks, the FPGA will give a better computational capability. Therefore, the utilization of DSP blocks of the FPGA is evaluated to determine the DSP efficiency (GOPS/DSP) for real-time classification. ZCU 102 SoC again have shown an efficient used of DSP blocks for real-time execution with 0.4 GOPS/DSP with having a total of 1280 DSP blocks.

Finally, the latency of the FPGA is the final metrics to be evaluated. The latency measures the time delay between the input processing and output generation of the whole system. By having a small latency, a real-time FPGA-based SL classification prototype is able to provide instant feedback for execution. PYNQ Zynq Ultrascale (ZU) board have the lowest latency of 0.85ms which shows the FPGA that executes SL classification task the fastest in real-time [103].

In overall, FPGA proves a suitable platform for CNN deployment due to its capabilities that tackles the needs of CNN and in realizing the application-specific integrated circuit (ASIC) application as a prototype. Besides, in according to the best FPGA for a real-time SL classification with overall high performing and low power consumption FPGA is the ZCU 102 SoC board. This is due to the efficient low power and energy consumption while utilizing the most numbers of operations and DSP blocks for a real-time execution.

Table 2.7
FGPA Hardware Performance Evaluations

| Study | Type of FPGA | Clock (MHz) | Quantization Strategy | Power (W) | Performance (GOPS) | Power Efficiency (GOPS/W) | DSP Used | DSP Efficiency (GOPS/DSP) | Accuracy | CNN | Latency |
|---|---|---|---|---|---|---|---|---|---|---|---|
| [8] | Zynq XC7Z020 | 150 | 8 fixed FP | 3.5 | 84.3 | 24.08 | 571 | 0.147 | - | - | - |
| | Arria 10 GX1150 | 240 | 8/16 float FP | 40 | 968.03 | 24.2 | 3136 | 0.308 | - | - | - |
| | VX980T | 150 | 8/16 float FP | 14.36 | 1000 | 69.63 | 3395 | 0.295 | - | - | - |
| | ZCU 102 SoC | 100 | 8 dynamix FP | 2.664 | 512 | 192.19 | 1280 | 0.4 | 99.26 | eCNN | - |
| [48] | PYNQ-Z2 ZC7Z020 | 100 | 8-bit fixed FP | 2.136 | - | - | 134 | - | 98.87 | ResNet18 | - |
| | | - | Partial 1-bit Binarization | - | - | - | 192 | - | 98.8 | Improved LeNet-5 | - |
| | | - | 1-bit Binarization | - | - | - | 28 | - | 85 | BNN | - |
| | AX301 XC7ZA200 | 50 | 16-bit Fixed | - | - | - | 571 | - | - | CNN | - |
| [103] | PYNQ Zynq Ultrascale (ZU) | 100 | 16-bit fixed point at last layer | 3.63 | - | - | 42 | - | 92 | Quantized-CNN | 0.85 |
| | - | - | - | 4.5 | - | - | - | - | 86.3 | Floating Point CNN | 16.6 |
| | Zedboard | 100 | - | - | - | - | 125 | - | 92.8 | Fixed Point CNN | 23.5 |
| | xcku060 | 200 | - | - | - | - | 646 | - | 80 | Fixed Point CNN | 42.5 |
| | PYNQ-Z2 ZC7Z020 | 100 | - | 1.8 | - | - | 28 | - | 85 | Binarized Networks | 1.2 |

## 2.5 Research Gaps and Limitations

Studies on developing a feasible convolutional neural network (CNN) digit sign language (SL) classification model prototype on field-programmable gate array (FPGA) consists of several research gaps and limitations that remains. These gaps need to be addressed to refine SL system's adaptability and feasibility.

One main key research gap exists between the software-optimized CNN model and its hardware implementation. Past research shows many developed CNN model for SL classification achieve high performance of more than 95% at the software level [5], [12], [13], [20], [21]. However, the optimized CNN lack considerations on the constraints for hardware implementation. To achieve high performance, CNN's architecture grows complex which demands large computational power and resources [8], [11], [14], [71]. This exceeds the capabilities of resource-constrained hardware platforms that suits for a portable and effective assistive device for the society. Consequently, there is a need for the co-design the development of CNN architecture design that associates with the consideration of the hardware's constraints for an effective implementation.

The hardware implementation challenge is significant for FPGA platforms. While FPGA have its ideal massive parallel processing capabilities that fit for CNN acceleration, its memory resources is limited which constrain the implementation of CNN model [8], [10], [11], [15]. This limitation is crucial for hardware implementation for it requires both computational efficiency and low power consumption for portable application-specific integrated circuit (ASIC) applications [6], [69], [74], [83], [102]. The trade-off between high performance and maintaining low power consumption remains a challenge in existing FPGA implementations [8], [17].

Another significant gap is the incorporation of pre-processing noise reduction filters in hardware. Noise reduction filter is recognized as an essential pre-processing method to enhance image quality and leads to improves CNN performance for classification task [13], [28], [85], [87], [88], [107]. Past research that utilizes the filter obtained clean input data that facilitates good feature extraction for the CNN model [85], [87], [88]. However, the clean input data presents an ideal environment to the filter where real-time environment is not considered. In real-time, the CNN with filter is implemented on hardware and the classification performance is low due to the lack consideration of hardware constraints which leads to hardware and software

performance gap [5], [8], [13], [16]. Moreover, there is still lack of comparative analysis on the performance of CNN with and without the filters to understand the filter's contribution and impact [13], [28].

Beyond the gaps, a broader limitation is the systematic approach of overall hardware-constrained CNN design that incorporates filters and its feasibility evaluation. Existing SL classification system often prioritizes either performance or efficiency but not altogether through hardware-software co-design approach for hardware implementation [8], [17], [18]. Furthermore, existing systems that achieve high performance are implemented on bulky and intrusive hardware prototype which is inconvenient for daily use [8], [17], [47], [108]. In hope of addressing the communication barriers, this poses another challenge for the deaf and hard of hearing (DHH) community and the society .

To address the interconnected gaps, a functional SL classification with feasible hardware prototype is required to implement the appropriate CNN model for resource-constrained FPGA that incorporates pre-processing filters through hardware-software co-design approach. This approach forms the basis focus of the present research.

# CHAPTER 3
# RESEARCH METHODOLOGY

## 3.1 Introduction

This chapter outlines the methodology to achieve the research objectives of developing a convolutional neural network (CNN) classification model prototype on a field-programmable gate array (FPGA) for digit sign language (SL) classification with a hardware-software co-design incorporation of filtering block approach. The proposed methodology is structured with each stages accomplishing respective research objectives. The study begins with developing a CNN classification model. The architecture of the CNN model used is a standard 7 layers CNN which is inspired by the existing CNN models like AlexNet, LeNet and VGG models [8], [51], [53], [76], [82], [109]. Then, the CNN model, without the application of filter, undergoes training, validation and testing in matrix laboratory (MATLAB) software with digit American SL dataset with no background noise which is widely available in Kaggle platform [19]. If the MATLAB evaluation is not satisfied and error, code debugging is conducted on CNN and undergoes training, validation and testing again. Once the MATLAB evaluation is accepted, the CNN is redesigned in Quartus according to the field-programmable gate array (FPGA) hardware compatibility. Then, the weights and biased of the trained CNN from MATLAB is converted to SystemVerilog hardware descriptive language (HDL) code module file using Python coding. The module file is exported to FPGA's software platform, Quartus, compiled with the rest of the module of the designed CNN. The modules are analyzed and synthesized, then programmed into the FPGA. With complete programming in FPGA, the same testing data set used in MATLAB is utilized to conduct real-time classification testing to compare the results. In case of errors during synthesize in Quartus and programmed in FPGA, code debugging is conducted to fix the problem. The results of CNN model without filter for training, validation and testing in MATLAB as well as real-time testing in FPGA are recorded as a benchmark result in the research. That marks the achieving of the 1$^{st}$ objective of the research.

Next, a pre-processing filtering block is introduced to the SL system to achieve the second objective. The incorporation of the filter block begins from the hardware implementation in Quartus. The idea of hardware-software co-design is to manage the system design based on computational constraints of the hardware and utilized the available FPGA pipeline [48], [106]. A module is created specifically for the filter block, and all other modules that interconnect with the pipeline are modified. Once the module is synthesized correctly and verified, the design of filter block in Quartus is adapted to the design of filter in MATLAB in achieving the realization of hardware-software co-design approach. Similarly to Quartus, the MATLAB file of the filter block is run to check for errors in the code. Failure of synthesise in Quartus and errors in MATLAB requires the conduct code debugging. Upon completion of the designed filter block file in MATLAB, the CNN model with the selected filter is run for training, validation and testing. After training, the weights and biased of the trained CNN model with filters are converted to SystemVerilog HDL. The final steps involve the conduct of hardware implementation testing. The steps are repeated with different selected filters applied in this research which are Gaussian, Median, Sobel and Bilateral filter.

Finally, the overall results of the CNN model with and without filters will be compared and evaluate to assess the feasibility of the CNN model implementation in FPGA, the impact of the pre-processing filtering block and finally determining the best filter for an enhance CNN model for digit SL classification. Figure 3.1 displays the overall flowchart of the proposed methodology. The proposed methodology potentially undergoes improvements based on the results of evaluation and testing. The refinement process involves in redesign and modifying parts or code of developing the system or retrain the CNN model to enhance the performance of the designed system.

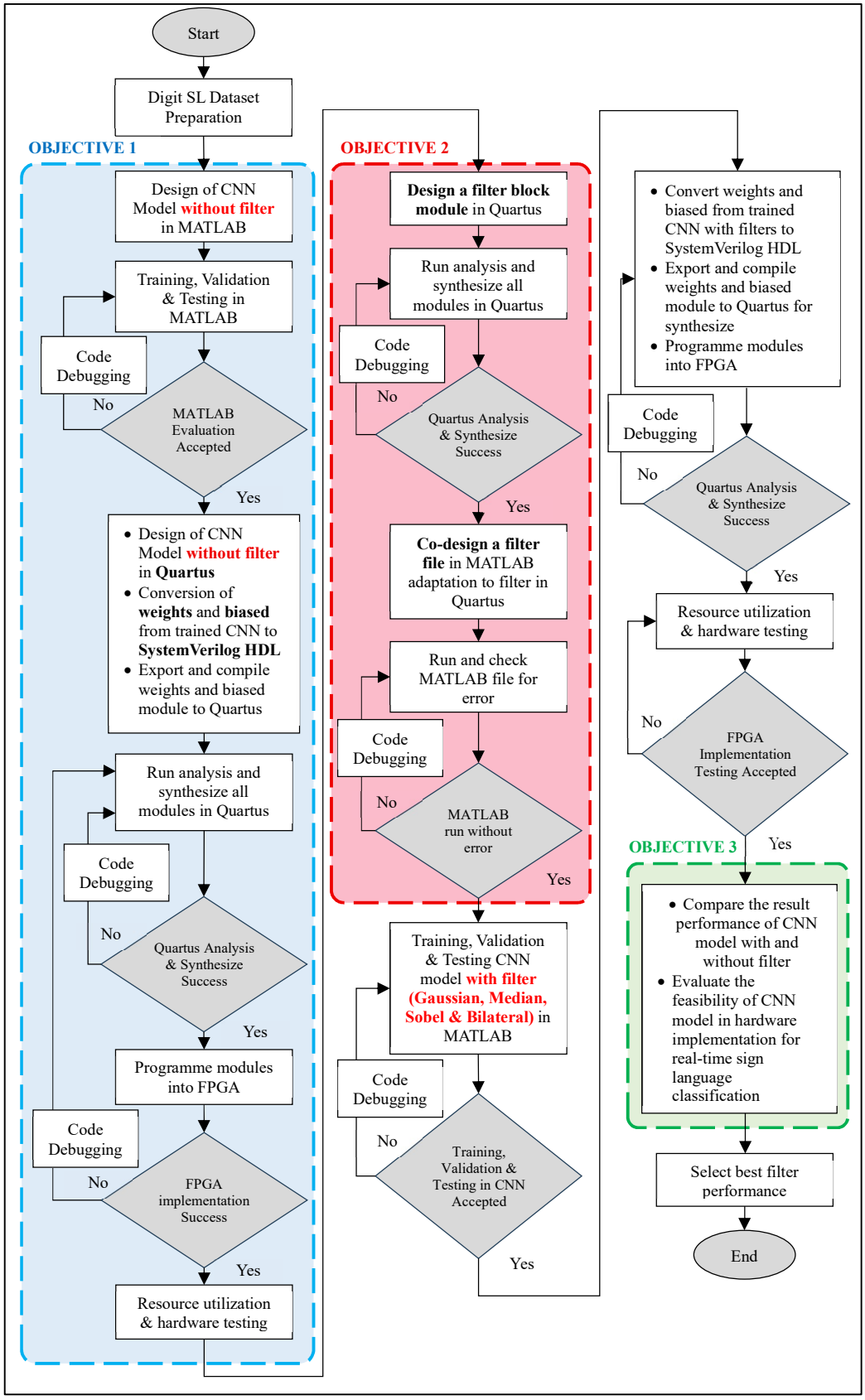



Figure 3.1 Overall Methodology For The Development Of CNN Classification Model Prototype On FPGA For Digit SL Classification

## 3.2 Digit SL Dataset Preparation

The digit American sign language (ASL) dataset is obtained from Kaggle website with controlled environment whereby no background noise [83]. The dataset consists of digit hand pattern of sign language (SL) which are easily identified and recognizable compared to dynamic gestures due to its static pose [21]. The samples of the digit ASL images are as shown in Figure 3.2.

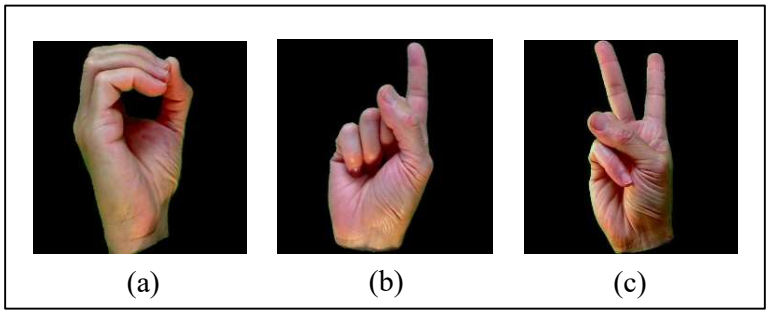



Figure 3.2 American SL Example Of Digit SL; Value 0(a), 1(b) and 2(c)

The selection of dataset serve as benchmark training of SL classification since the research focuses on the feasibility of the designed on convolutional neural network (CNN) for hardware implementation and the impact of co-design filters on CNN performance. As most SL dataset provided in Kaggle consists of combine alphabets, words and digit that requires many classes to classify, the selection of dataset is scoped down to digit only which have 10 class of classification. The dataset comprises of 5,000 samples of static digit SL, where each digit of 0 to digit 9 have 500 images each. The dataset is split into 70/15/15 ratio for training, validation and testing respectively [5], [110]. The data splitting helps in preventing overfitting, and generalization performance. The data is further prepared according to the flow presented in Figure 3.3.

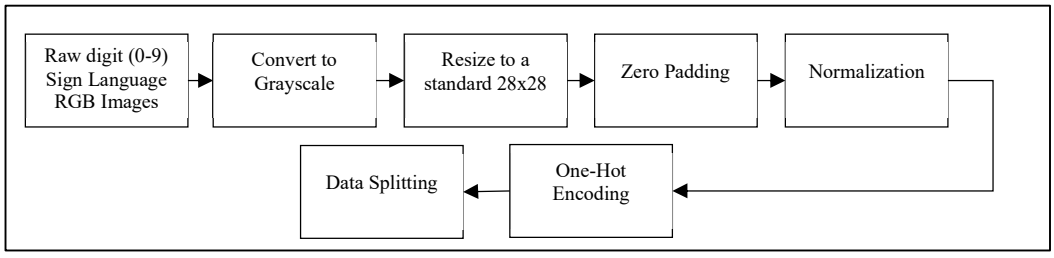



Figure 3.3 Data Preparation Flow

The data preparation flow begins with converting raw input images of the dataset from Red-Green-Blue (RGB) to grayscale. The conversion is essential as SL classification relies on shapes and edge information rather than the colours. At the same time, the conversion reduces the data volume of RGB that require 3 channels while grayscale only 1 channel. Hence, makes the CNN more computationally efficient.

After grayscale conversion, each image is resized to a standard 28x28 pixels using interpolation techniques to ensure a uniform dimension regardless the original size of the image. The standardization is required to maintain consistent weight matrices throughout the CNN. Then, the images are padded with 1-pixel zero padding as a border that creates a 30x30 dimension input before feeding it to the CNN. The zero padding have critical purposes for CNN which are to preserve the spatial dimensions through convolution operations, ensuring border pixels are extract equally as centre pixels for feature extraction and maintain mathematical relationships between layers. Without zero padding, there are potential loss of edge information at the border pixel before deeper layers can process them.

After that, the pixels are normalized to [0,1] range by dividing all pixels' value by 255 (grayscale gradient) to prevent any single pixel to dominate the gradient updates during training. The normalized images are stored in training data tensor as single-precision floating-point values to optimize the memory usage and computational speed. At the same time, labels are one-hot encoded for classification and stored in training labels. This transforms each digit (0-9) into a 10-position vector with a single '1' at the right digit's position and '0s' elsewhere (e.g., digit '5' becomes[0,0,0,0,0,1,0,0,0,0]). The transform prevents the CNN to interpret the digit as numerical values and treat them equally as different category for classification. The final image's size will be 30x30 pixels to complement the convolutional kernel size of 3x3 pixels in the network. This marks the completion of data preparation and ready to feed into CNN.

## 3.3 CNN Model Development in MATLAB

In this stage, the focus is to design convolutional neural network (CNN) model for sign language (SL) classification. The design of the CNN classification model is conducted in matric laboratory (MATLAB) software. CNN models such as AlexNet, LeNet and VGG are taken as reference to the design of the CNN classification model of this study [8], [51], [53], [76], [82], [109]. These models are selected due to its success in classification task of other fields. This sets a benchmark for the designated CNN model in achieving a functional CNN for SL classification. Finally, the CNN model undergoes training, validation and testing. The evaluation of the CNN lies on the performance throughout training and the results of the required metrics obtained. The evaluation is satisfied once all metrics are achieved without error in the codes of MATLAB during training. The metrics used to evaluate the CNN performance are elaborated in detail in Section 3.6

### 3.3.1 Design of CNN Model Architecture

The designed 7-layer convolutional neural network (CNN) architecture for this research is as shown in Figure 3.4. In the first layer of the architecture, a 3x3 convolutional kernel is applied to the input image of 30x30 pixels to generate 28x28 pixels with 16 feature maps and this process is called convolution. The convolution operates by reducing the spatial dimensions according to the kernel size (3x3).

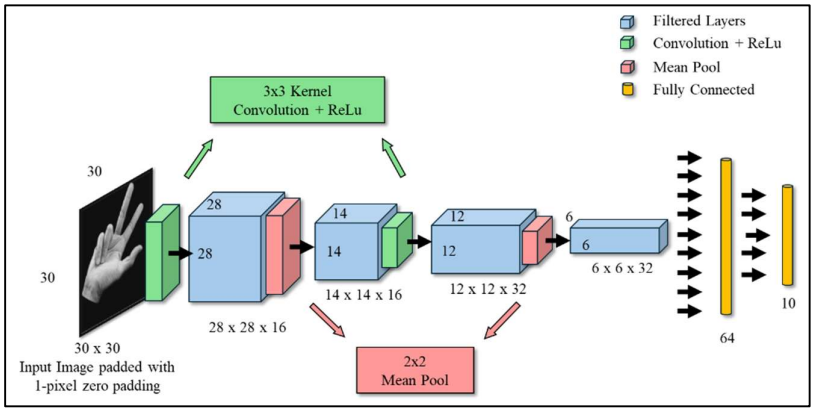


Figure 3.4 7- Layered Network CNN Architecture

The output dimension from the convolution is calculated based on the input size and kernel width. For each of the 16 output feature maps, CNN creates kernels that connect to the single grayscale channel. The kernel weights are initialized with random values symmetrically distributed around zero (-0.25,0.25). This ensures a diverse feature detection while maintain stable activations through the CNN. Then, the weights are quantized to a 16-bit fixed point representation to reduce memory requirements while preserve precision.

After convolution, the rectified linear unit (ReLu) activation function is applied by adding nonlinearity to the model. The non-linearity allows CNN to learn complex patterns. The formula of ReLu activation function is shown in (3.1).

$$f(x) = \max(0,x) \tag{3.1}$$

Where :

f(x) = Output value after applying ReLu

0 = Constant use to compare with value "x"

x = Input value from previous layer into the activation function

The biases in the 1st convolutional layer with ReLU are initialized to zero for each feature map which serves as learning parameters that is adjusted during training. Then, a 2x2 mean pooling layer is applied to reduce the spatial dimensions of the feature map from 28x28 pixels to 14x14 pixels by averaging values in each 2x2 region. The mean pool reduces computational load by decreasing parameter count for subsequent layers. At the same time, it accumulates local pixels and turns into more robust feature. Mean pooling emphasizes smooth feature representation compared to max pooling by averaging neighbourhood pixels rather than selecting extreme pixel values. This preserves the important shape characteristic for distinguishing similar digit sign language (SL) pose.

The data convolves again in the 2nd convolution layer with 32 filters of size 3x3. In this layer, all 16 feature maps from the 1st mean pool layer as input channels. The weight initialization creates kernels connecting each of the 32 output feature maps to all 16 input channels that produces hundreds of individual kernels. Each kernel is done similarly to the initialization made in the 1st convolution layer, followed with the 14-bit quantization. The filters now operate on patterns that combines edges and shapes from

the 1st convolutional layer which allows detection of higher-level structures such as curves and intersections according to the digit SL pose. The convolution operation on 14x14 input produces 12x12 feature map with 32 input channels for the 2nd mean pool layer. The 2nd mean pool layer reduces the size of the feature maps from 12x12 to 6x6 pixels where each location of the 6x6 grid corresponds to a region of the initial image that has undergoes 2 convolution and 1 mean pool. This gives a large receptive field that represents most of the original image, enabling a recognition of a holistic pattern.

The output from the 2nd mean pool layer is then flattened in the 1st fully connected (FC) layer with ReLU. The 1st FC layer contain 128 nodes that each connects to 1152 inputs. The weight matrix connecting the layers are initialized the same way in the 2 convolution layers, with biases initialized similarly followed by the 14-bit quantization. Each of the 128 neurons connects to every element of the input vector which allows learning global patterns. This is where the CNN synthesizes all extracted features by previous layers and combining information of shapes to form a complete digit SL representation.

Finally, at the 2nd FC layer, the 128 features from 1st FC layer are mapped to 10 output nodes that represents the 10 possible digit SL classification. This marks the final process of the CNN and complete the classification.

### 3.3.2 CNN Model Training Process in MATLAB

The designed 7-layer convolutional neural network (CNN) model is trained, validated and tested using the prepared digit American sign language (SL) dataset. In this stage, the training process utilizes the training dataset from the data split and its one-hot encoded labels to adjust the weights and biases through forward propagation, loss calculation, backpropagation, and gradient descent. Throughout training, the CNN's performance is periodically evaluated on the validation dataset to prevent overfitting and monitor CNN's generalization.

In training, the process begins with the training dataset divided into batches of 5 images each. This process is called a mini-batch training. Then, several major processes are performed during the training to optimize the weights and biases of the CNN model. The first major process is called forward pass. The forward pass function

performs by propagating the input batch through the network to calculate the activations for each layer that ultimately generate predictions from the output layer.

The second major process is the loss calculation. This process stored and computed all the errors for each batch by comparing the predictions to the true predictions. Two loss function are used depending on the activation function of the output layer. When sigmoid or softmax activation is used, $L_C$, cross-entropy loss is applied. The formula of cross-entropy is shown in (3.2).

$$L_C = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)] \quad (3.2)$$

Where :

$L_C$ = Cross-entropy loss

N = Number of samples in a batch

$y_i$ = True label for sample *i*

$p_i$ = Predicted probability of class 1

*i* = i-th sample

The sigmoid activation function outputs values between 0 and 1 which makes it suitable for binary classification, while the softmax activation function extends the output values to multi-class classification by producing probability distribution across all classes.

On the other hand, when tanh activation is used, $L_Q$, quadratic loss is applied. The tanh activation function outputs values between -1 and 1 which is suitable with quadratic loss for regression-like outputs. The formula of quadratic loss is shown in (3.3).

$$L_Q = -\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (3.3)$$

Where :

$L_Q$ = Quadratic loss

$\hat{y}_i$ = Predicted output for sample *i*

Next process is the backpropagation. This process computes gradients by propagating on the error backward through the CNN. The function propagates error back through each layer to compute gradients with respect to the weights and biases at each layer. Finally, is the gradient descent. The gradient descent updates the weights

based on the calculated gradients and learning rate, iteratively to minimize the overall error.

### 3.3.3 CNN Training Configuration and Hyperparameter Tuning

The hyperparameters for training are fined tuned to the following values in Table 3.1. At the same time, the hardware configuration of CNN used for training includes the central processing unit (CPU), graphics processing unit (GPU) and random-access memory (RAM) details include in Table 3.1. The hyperparameters including learning rate, number of epoch and batch sizes are maintained throughout training to control the behaviour of the model consistently throughout the research. This ensures a fair comparison between the benchmark model with the filter-applied CNN model.

Table 3.1
CNN Configuration and Hyperparameter Tuning for Training

| Hyperparameter | Value |
|---|---|
| CPU | Intel Core i7-14700F |
| GPU | NVIDIA GeForce RTX 5060 Ti 16GB* |
| RAM | 32GB (2x16GB) 6000MHz |
| Learning rate | 0.01 |
| Number of Epoch | 100 |
| Number of Batches | 5 |

## 3.4 CNN Model Implementation on Field-Programmable Gate Array (FPGA)

This section details the hardware implementation of the trained convolutional neural network (CNN) model onto field-programmable gate array (FPGA) platform for real-time digit sign language (SL) classification. The implementation covers the design flow of the construction of CNN architecture in Quartus for FPGA implementation using SystemVerilog, followed by the hardware configuration setup of the D8M-GPIO camera and Terasic DE2-115 FPGA board. The internal pipeline of FPGA connecting the CNN with the camera interface and display modules is presented. Subsequently, the conversion process of 32-bit float points weights and biases from trained CNN model in matrix laboratory (MATLAB) into 16-bit fixed point format that compatibles with the FPGA environment is elaborated. Finally, the FPGA evaluation of real-time testing and resource utilization metrics are outlined later in Section 3.6.

### 3.4.1 CNN Architecture Design in Quartus

In hardware implementation on field-programmable gate array (FPGA), the CNN architecture is redesigned using Quartus, FPGA's programming software, to create a SystemVerilog hardware description language (HDL) version of the CNN. This is necessary due to the FPGA unable to execute matrix laboratory (MATLAB) code directly and utilize the CNN structure in MATLAB. Therefore, The CNN architecture is reconstructed by replicating the CNN layer's functionality in MATLAB, including convolutional layers, mean pooling layers and fully connected layers. The CNN in FPGA does not perform any training but executes inference using the converted weights and biased from the trained CNN model in MATLAB.

The complete design of CNN in Quartus integrates FPGA hardware implementation and its interface modules. The system architecture consists of three main modules in Quartus which are the CNN, camera interface for image acquisition and display of output visualization. The CNN accepts input 30x30 pixel grayscale images and produces 10-class of digit SL prediction similar to MATLAB. To interface with real-time camera, additional modules are incorporated into the system design through the internal pipeline architecture. The FPGA internal pipeline involves the design of interconnected modules that complements one another for real-time digit SL classification. The connection of the modules is as shown in Figure 3.5.

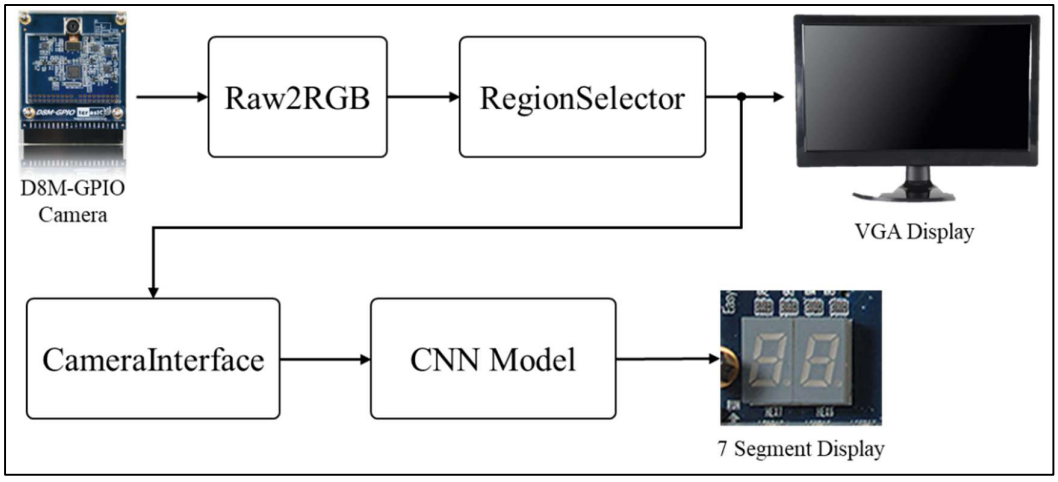



Figure 3.5 FPGA Interconnected Modules

1. Raw2RGB : When the image is captured, the raw input image from camera is converted into RGB values.

2. RegionSelector : The RGB values are enhanced with contours drawn around a specific area in a graphical display system. The contours are determined by the pixels value and a threshold that creates a bounding box in the video graphic array (VGA) monitor that guides users to focus the image sensor on the target object.
3. CameraInterface : The data output of the RegionSelector is processed in the form pixels and sends to CNN model for classification. CameraInterface block is designed to interfaces the camera with the CNN for digit SL classification. This helps the transferring and conversion of data compatible for the whole system.
4. CNN Model : The data goes to 7-layered CNN model for classification and send the classification decision result to the 7-segment display.
5. 7 Segment Display: Displays the final output of the classification made from the CNN model and determine the real-time performance of the whole system for SL classification.

### 3.4.2 FPGA Hardware Configuration Setup

The field-programmable gate array (FPGA) based digit sign language (SL) classification system utilizes two main hardware components which are the D8M-GPIO camera for image acquisition and Terasic DE2-115 FPGA development board as the main processing platform for. The specifications of the components are tabulated in Table 3.2 and Table 3.3 respectively.

Table 3.2
D8M-GPIO Camera Specifications

| Component | Specification |
|---|---|
| Sensor Chip | OV8865 (8 Megapixel) |
| Output Format | 10-bit parallel Bayer pattern |
| Interface | 2×20 GPIO with 3.3V I/O |
| Maximum Resolution | 3264 × 2448 pixels |
| Frame Rate | 60 FPS (1408×792), 30 FPS (3264×2448) |

Table 3.3
Terasic DE2-115 FPGA Development Board Specifications

| Component | Specification |
|---|---|
| FPGA Chip | Altera Cyclone IV E (EP4CE115) |
| Logic Elements (LEs) | 114,480 LEs |
| Memory | 128 MB SDRAM, 2 MB SRAM, 8 MB Flash |
| Clock Sources | 50 MHz oscillators |

The hardware configuration consists of D8M-GPIO camera, Terasic DE2-115 FPGA development board and video graphic array (VGA) monitor to display the real-time capture of the camera. The hardware configuration is as shown in Figure 3.6. The D8M-GPIO camera and VGA monitor requires internal pipeline configuration to align with the connections within the FPGA.

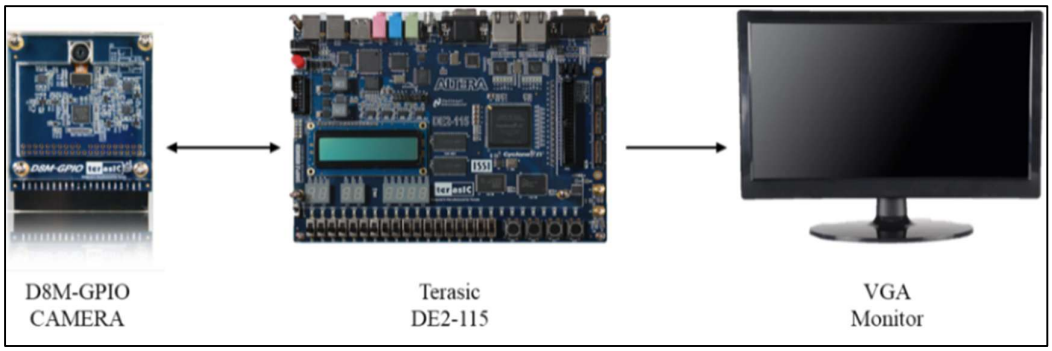



Figure 3.6 FPGA Programme Setup Configuration

The selected hardware for digit SL classification is the Terasic DE2-115 FPGA development board. This is due to its parallel computational processing for efficient performance and low power consumption which suits the realization of application-specific integrated circuit (ASIC) application [8], [10], [11], [96], [104].

The hardware implementation of CNN requires the programming FPGA. The FPGA programming begins with the software configuration involving Python software to convert the trained CNN model's weights and biased data stored in MATLAB into SystemVerilog hardware descriptive language (HDL) for Quartus, which is a software for FPGA programming. The python file generates the corresponding files to compile with the modules of designed CNN in Quartus software for synthesizing and verification of files.

After the conversion process is complete, the Quartus is synthesized to program the FPGA for real-time digit SL classification. A real-time test is conducted using the same testing dataset used in MATLAB. The images of the test set are displayed via mobile phone screen for the camera to capture and classify in real-time. This approach is conducted in a low light room with the mobile phone's brightness set to maximum for the FPGA to completely focus on the test image displayed. A sample look at how the real-time classification is conducted is shown in Figure 3.7, but the real classification is done in a darker room than presented.

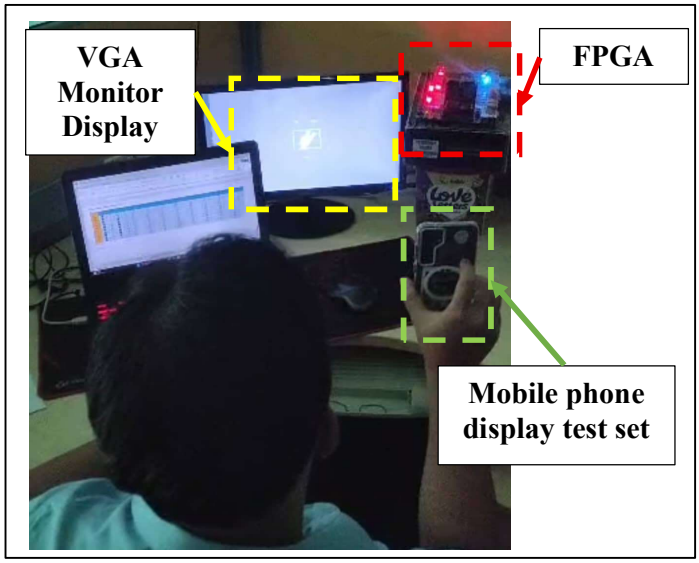


Figure 3.7 Sample Situation Conducted For Real-Time Classification

The approach validates the FPGA implementation produces the same classification results as the MATLAB testing when presented the same testing dataset. The feasibility evaluation of the hardware prototype depends on the real-time digit SL classification of the FPGA and the resource utilization of the CNN model in FPGA. The FPGA displays the prediction classification on the 7-segment display and the outcome is evaluated. If the hardware is unable to recognize SL, the internal connection of SystemVerilog files will be further refined. Moreover, if the hardware successfully recognizes SL but underperforms for digit SL classifications, the CNN model requires further co-design enhancement to compensate with the hardware constraints.

### 3.4.3 Weights and Biased Conversion Process Configuration

The software configuration in this research utilizes 3 platform which are MATLAB, Visual studio and Quartus and the flow of configuration is as shown in Figure 3.8. This is configuration is reserve for the conversion of the weights and biased from the trained convolutional neural network (CNN) model which is ready for field-programmable gate array (FPGA) implementation. The conversion process translates the 32-bit floating-point weights and biased in MATLAB into 16-bit fixed-point binary representations to be stored in FPGA memory blocks. This is due to FPGA readability unable to interpret the floating-point format in MATLAB. With the conversion, the weights and biased values are preserved for accurate inference in FPGA.

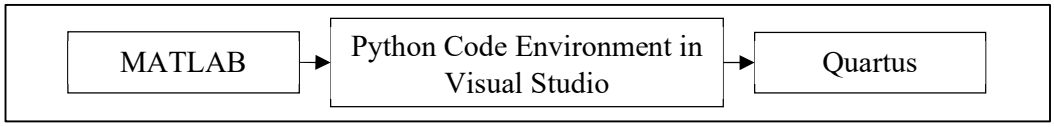



Figure 3.8 Software Used For Weights And Biased Conversion Process Configuration

The configuration begins with the complete development of CNN model that successfully undergoes training, validation and testing within the MATLAB. Once the CNN model is trained, the weights and biased script files are generated and stored in MATLAB which will be used for the python code environment in visual studio. Then, a python file is created in the visual studio platform to generate SystemVerilog hardware descriptive language (HDL) memory file modules for storing the weights and biases values into FPGA. The python file helps to adapt the MATLAB-trained CNN model to FPGA implementation by converting the weights and biases into memory using interleaved distribution that suits for synthesis in Quartus. Finally, the code files are synthesized in Quartus and programmed the system into FPGA for a real-time application.

## 3.5 Enhancement of CNN Model Performance Using Hardware-software Co-design Approach

In this stage, the focus is to enhance the convolutional neural network (CNN) model's performance by incorporating an image filtering block into the field-programmable gate array (FPGA) pipeline to compare the results with the benchmark results of CNN model without filter. The filters provide cleaner and more detailed input data which facilitates the CNN with better feature extraction and potentially improved its classification accuracy [85], [87], [88]. The same training, validation and testing process in matrix laboratory (MATLAB) software is repeated but with the input images pre-processed with the corresponding filters before feed into the CNN. This ensures the trained model learns from filtered images that match the FPGA's processing during real-time inference. After training, validating and testing of train each CNN model with an applied filter, the performance of each filter-applied CNN model is evaluated and compared with the benchmark results of the CNN model without filter. The evaluation

metrics are similar to the metrics acquired for the benchmark CNN model without filter which is further elaborated in Section 3.6.

The idea of the co-design is to manage the system based on the hardware constraints and utilized the available FPGA's pipeline [48]. As discussed in section 2.3.5, an effective co-design understands the available memory and processing elements before designing the CNN [8], [15], applying a quantization strategy between hardware and software [48] as well as enable pipeline with on-chips memory sharing between filter and CNN [17], [18].

### 3.5.1 Incorporation of Co-design Pre-Processing Filtering Block

To enhance the performance of the convolutional neural network (CNN) model, an additional block of an image filtering block is incorporated into the system. In matrix laboratory (MATLAB) environment, the filter block is commonly be applied to during the data preprocessing. However, this research uses a hardware-software co-design approach where the filter block is designed and implemented in the hardware pipeline of field-programmable gate array (FPGA) first. This ensures the real-time operates at 60 frame per second (fps) and ensures MATLAB training pipeline reflects the hardware behaviour correctly. At the same time, enables efficient resource utilization through the sharing of on-chip memory between the filter block and CNN [17], [18].

The co-design approach considers the FPGA's hardware constraints and utilizes available resources while ensuring the filter and CNN-based digit sign language (SL) system in Quartus operates in real-time [8], [15], [17], [18]. Several FPGA constraints influenced the co-design of the filter. One of the constraints is the limited digital signal processing (DSP) blocks which cause gaussian filter to use sequential multiplication instead of parallel multipliers. Besides, the limited memory block also causes restrictions of all filters to a 3x3 window size that uses two-line buffers. Next, the 50MHz clock of FPGA necessitates multi-stage sorting especially for the median filter and state-machine approaches across all implementations. Finally, with the need of border handling, additional logic is required for boundary detection, and the concern of resource sharing cause the optimization of bilateral filter with a lookup table for range weights [48]. The design code of filters in FPGA is presented in .

The hardware constraints lead to the co-design of filtering block which is incorporated to the FPGA interconnected modules after the camera and Raw2RGB

block as shown in Figure 3.5, before feeding the image to the RegionSelector block and CameraInterface, finally the CNN block as shown in Figure 3.9.

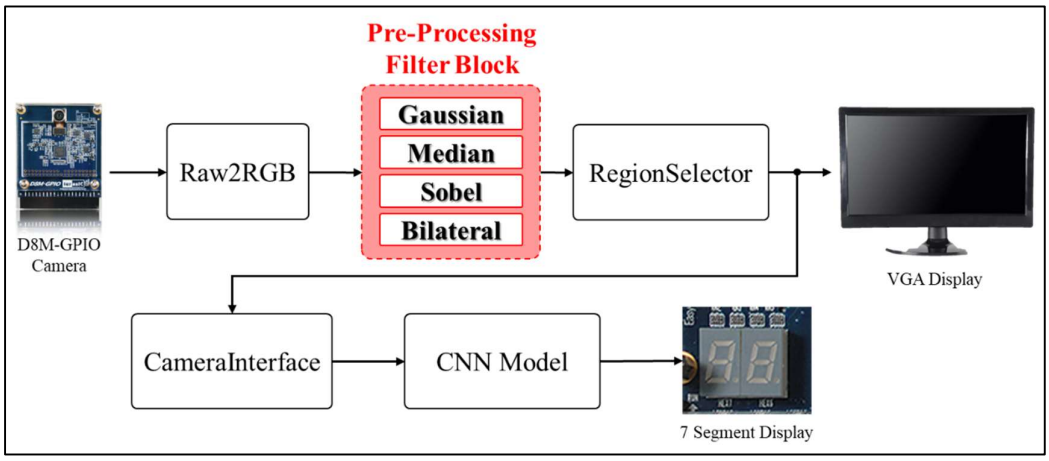



Figure 3.9 Incorporation of Co-design Filtering Block

The following are filters implemented within the co-design filtering block in the FPGA pipeline :

i. Gaussian Filtering [7], [111], [112] : Gaussian filter applies a 3x3 kernel weights with integer weights to smoothen the input images. The filtering operation formula is shown in (3.4).

$$G(x,y) = \left(\frac{1}{16}\right)\sum_{i=-1}^{1}\sum_{j=-1}^{1}[I(x+i,y+j) * K(i+1,j+1)] \quad (3.4)$$

Where :

G(x,y) = Output pixel value at coordinate (x,y)

I(x,y) = Input image pixel value at coordinate (x,y)

K = Gaussian kernel with integer weights of [1,2,1; 2,4,2; 1,2,1]

i , j = Offset indices ranging from -1 to 1

The kernel approximates a gaussian distribution to give the highest weight to the centre pixel of image and decrease weights to neighbouring pixels. The filter helps to reduce a high-frequency noise in the image by averaging pixel values with neighbour pixels.

ii. Median Filtering [7], [42], [112] : Median filter is used to reduce salt-and-pepper noise of the input image. The filtering operation formula is shown in (3.5).

$$M(x,y) = median\{I(x+i,y+j)\}, \quad for\ i,j \in [-1,0,1] \quad (3.5)$$

The filter operates by replacing pixels with the median of corresponding pixels which effectively remove the outliers. The filter preserves edges while removing noise, making it suitable for digit SL hand gesture.

iii. Sobel Edge Detection [13], [102], [109], [111] : Sobel highlights the important edges of the input image by computing the intensity gradient using sobel operators. The gradient magnitude is computed as shown in (3.6).

$$G = |G_x| + |G_y| \tag{3.6}$$

Where :

$G_x$ = I*$K_x$

$G_y$ = I*$K_y$

With operation kernels as shown in (3.7).

$$K_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \; K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{3.7}$$

The filter emphasizes edges by computing both horizontal and vertical gradients using kernels. The computation of both gradients is conducted in many stages and then, the gradient magnitude is approximated. This produces the edge map where pixel intensity corresponds to the highlighted edge.

iv. Bilateral Filtering [74], [92], [93] : Bilateral filter smoothens input image while keeping the edges sharp by considering the intensity similarity and spatial proximity. The filtered output is defined as shown in (3.8).

$$BF(x,y) = \frac{1}{W}\sum_{i=-1}^{1}\sum_{j=-1}^{1}[I(x+i,y+j) * w_s(i,j) * w_r(\|I(x+i,y+j) - I(x,y)\|)] \tag{3.8}$$

Where :

BF(x,y) = Output pixel value at coordinate (x,y)

$w_s$ = Spatial weight based on distance from centre pixel

$w_r$ = Range weight based on intensity difference

W = Normalization factor

The filter combines spatial weights using gaussian approximation with range weights which determines by the difference of pixel intensity. Furthermore, the filter computes the weighted sum of the high receiving weight pixels that have similar intensity to the centre pixel. The range weights are obtained using a lookup table that assigns the high to low intensity differences.

### 3.5.2 Adaptation of Co-design Filtering Block in MATLAB

Following the co-design approach in field-programmable gate array (FPGA), the designed filter block is adapted and validated in matrix laboratory (MATLAB) to ensure its compatibility for training [48], [96]. This adaptation involves incorporating the software design of the same filters designed in FPGA that are applied to the training dataset in MATLAB during data preprocessing. The MATLAB implementations of filters are co-designed to exact mirror of the behaviour of the filters in the FPGA pipeline.

The co-design approach ensures the method of designing the filters considers the FPGA's hardware constraints, such as the lookup table for range weights used in bilateral filter and the multi-stage sorting network for median filter [17], [48]. The approach makes sures the filters behave similarly in both MATLAB and FPGA despite the differences of execution [11], [48]. This makes the CNN learns from the filtered data in the same way during real-time inference.

For gaussian filter, the MATLAB implementation uses the same 3x3 kernel with weights [1,2,1; 2,4,2; 1,2,1] which are normalized by the factor of 1/16. At the same time, the input image is padded by replicating border pixels in handling the edges.

However, for median filter, the each 3x3 window are extracted, then are flattens the pixel values into a single array. The array is sorted in ascending order, and the $5^{th}$ element is selected as the median value. The border padding is also replicated to ensure that the filter operates correctly at the image edges.

On the other hand, sobel filter applies both horizontal and vertical gradient kernels to each 3x3 window matching the FPGA implementation, also with the gradient magnitude to be approximated as $|G_x|+|G_y|$. The results are clamped to the 0-255 grayscale pixel range.

Finally, the bilateral filter follows the two-weight approach. The spatial weights are combined with the range weights that determines by the lookup table based on the intensity difference between centre and neighbouring pixels. The lookup table assigns the decreasing weight values as the intensity difference increase. The weight values also range from 255 for identical pixels to 0 for differences greater than 50. The final output will be the sum of the normalized weight.

The co-design of each filter in MATLAB is presented in Appendix 1 for further look. The adaptation verifies the co-design is done correctly and ensures its compatibility for convolutional neural network (CNN) model with applied filter training in MATLAB. With the same behaviour of design filters in both MATLAB and FPGA, the CNN learns from the filtered data similarly.

Once the co-design filter is verified and runs without error in MATLAB, then each CNN model with applied filter undergoes training, validation and testing. The performance of each CNN is recorded and compared against the benchmark model without filter. This determines the impact of incorporating pre-processing filters in improving the CNN performance for digit sign language (SL) classification. The results of the trained models in MATLAB are tabulated, and their weights and biases are converted for FPGA implementation which further used for real-time test evaluation.

## 3.6 Overall Evaluation of CNN Models with and Without Filters

This section presents the evaluation metrics for the overall performance of convolutional neural network (CNN) models with applied filters and the benchmark CNN model without filter. The same metrics are consistently applied across training, validation and testing in matrix laboratory (MATLAB) as well as the field-programmable gate array (FPGA) evaluation for fair comparison.

### 3.6.1 Evaluation Metrics for Training and Validation in MATLAB

During training and validation in matrix laboratory (MATLAB), four fundamental metrics are being focused for the evaluation of CNN model performance. The metrics are training and validation accuracy, precision, recall and F1-score. The metrics are elaborated as follows:

a) Training: Computes the accuracy of the CNN model on the training dataset during. The accuracy is computed by the number of correct predictions divided by the total predictions as shown in (3.9).

$$\text{Training Accuracy} = \frac{Number\ of\ Correct\ Predictions}{Total\ Predictions}\ x\ 100\% \tag{3.9}$$

b) Validation Accuracy : Similarly, computes the accuracy of the CNN model like training but on the validation dataset after training. The accuracy is computed is shown in (3.10).

$$\text{Validation Accuracy} = \frac{Number\ of\ Correct\ Predictions}{Total\ Predictions}\ x\ 100\% \tag{3.10}$$

c) Precision : The proportion of positive predictive values that are correct indicates the model's reliability when it predicts a positive class [12], [13]. The formula for precision is shown in (3.11).

$$\text{Precision} = \frac{True\ Positives(TP)}{True\ Positives(TP) + False\ Positives(FP)}\ \text{x}\ 100\% \tag{3.11}$$

Where :

True Positives (TP) = Correct predicts positive class

False Positives (FP) = Incorrect predicts positive class when true class is negative

d) Recall : The proportion of actual positives that are correctly predicted indicating the model's ability to find all positive value [12], [13]. The formula for recall is shown in (3.12).

$$\text{Recall} = \frac{True\ Positives(TP)}{True\ Positives(TP) + False\ Negatives(FN)}\ \text{x}\ 100\% \tag{3.12}$$

Where :

False Negatives (FN) = Incorrectly predicts a negative class when true class is positive

e) F1-score : The harmonic means , which in terms called F1-score, is the balance trade-off between the precision and recall [12], [13]. The formula for F1-score is as shown in (3.13).

$$\text{Harmonic means, F1-score} = \frac{2\ X\ Precision\ X\ Recall}{Precision + Recall}\ \text{x}\ 100\% \tag{3.13}$$

### 3.6.2 Evaluation Metrics for Testing in MATLAB

After the completion of training, the CNN model is evaluated on the unseen testing data using the same metrics in training and validation which are precision, recall, F1-score. There are also additional 3 fundamental metrics which are testing accuracy, confusion matrix and inference speed as follows:

a) Testing Accuracy : Computes the accuracy of the CNN model similarly like training and validation but on the unseen testing dataset after the completion of training. The testing accuracy calculation is as follows in (3.14).

$$\text{Testing Accuracy} = \frac{Number\ of\ Correct\ Predictions}{Total\ Predictions}\ x\ 100\% \tag{3.14}$$

b) Inference speed : The average time required to process a single sample through the trained model indicating the computation efficiency and real-time capability. The formula for inference speed is as shown in (3.15).

$$\text{Inference speed} = \frac{Total\ inference\ time}{Number\ of\ samples} \tag{3.15}$$

c) Confusion Matrix : An error analysis that revels specific confusion patterns and misclassification made by the trained CNN model, displays the insight into the classification performance. The off-diagonal entries within the confusion matrix represents the mislabelled image, while the sum of diagonal values divided by the total samples determines the overall classification accuracy. An example of confusion matrix is as shown in Figure 3.10.

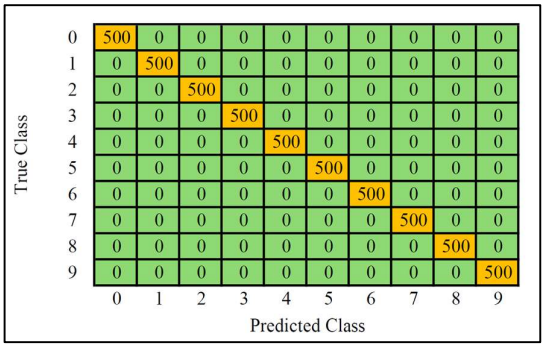



Figure 3.10 Example of an Ideal Confusion Matrix Results

### 3.6.3 FPGA Prototype Real-time Testing Evaluation

For the field-programmable gate array (FPGA) prototype, additional metrics which are specifically for hardware evaluation are used as well as the classification performance metrics described in matric laboratory (MATLAB) testing. The FPGA evaluation focuses on real-time performance using the following metrics :

a) Testing Accuracy : Same accuracy metric in Equation that uses the same testing dataset which displayed via mobile phone for real-time classification.
b) Resource Utilization : Total number of utilized logic elements (LEs) and registers in FPGA based on the compilation report after synthesis. The evaluation of these 2 resources is crucial in evaluating the impact of FPGA implementation of CNN.

- Logic Elements (LEs) : The total number of LEs indicates the computational cost of the implemented CNN model.
- Registers : The total number of registers indicates the timing and state management requirements of CNN model.

The compilation report in FPGA gives insights into the overall performance of the implemented CNN-based sign language (SL) classification system. The results of resource utilization and classification performance determine the feasibility of FPGA prototype for real-time digit SL classification.

### 3.6.4 Comparative Analysis of Performance Improvement

The final evaluation conducts a comparative analysis of the convolutional neural network (CNN) performances with and without filters using all the metrics defined in Section 3.6.1 through Section 3.6.3 that covers the evaluation in matrix laboratory (MATLAB) and field-programmable gate array (FPGA) prototype. The comparison of results determines the feasibility of CNN for digit sign language (SL) classification in FPGA and the impact of co-design filters in improving the CNN's performance. The results of the comparison represent the performance improvement of the CNN models without filters over the benchmark model. The improvement is computed using the formula of relative performance improvement as shown in (3.16).

$$Improvement = \frac{M_{filter} - M_{benchmark}}{M_{benchmark}} \text{ x } 100\% \quad (3.16)$$

Where :

$M_{filter}$ = Metric value for the CNN model with filters

$M_{benchmark}$ = Metric value for the benchmark CNN model without filter

The relative performance improvement calculation give insight into the enhancement achieved by incorporating the pre-processing filter to the CNN for digit SL classification. Positive results of the calculation show the model with filters outperform the benchmark mode. While negative results show the degradation of performance.

On the other hand, the same formula is applied for the relative increase in logic elements and registers utilization during FPGA implementation. The formula signifies the additional resource utilization as percentage of the baseline resource consumption and the formula is expressed as shown in (3.17).

$$Resource\ Increase = \frac{R_{filter} - R_{benchmark}}{R_{benchmark}} \text{ x } 100\% \quad (3.17)$$

Where :

$R_{filter}$ = Resource utilization for the CNN model with filters

$R_{benchmark}$ = Resource utilization for the benchmark CNN model without filter

# CHAPTER 4
# RESULTS AND DISCUSSION

## 4.1 Introduction

In this chapter, the final evaluation of the convolutional neural network (CNN) model on field-programmable gate array (FPGA) for digit sign language (SL) classification is conducted. The evaluation results will be based on the performance of 5 trained CNN models which differentiate by without filter, with gaussian filter, median filter, sobel filter and bilateral filter as configured in the methodology. This chapter embodies three main sections that align with the research objectives. The first section presents the complete evaluation of benchmark CNN model without filter from matrix laboratory (MATLAB) training to FPGA implementation. The next section covers the complete evaluation results of all four CNN models with respective applied filters in MATLAB and FPGA. The last section discusses a comparative analysis between the CNN models evaluated throughout the research. The experimental setup and evaluation metrics in methodology are applied here.

## 4.2 Overall Performance of CNN Model Without Filter

This section presents the complete evaluation of the convolutional neural network (CNN) model without the incorporation of pre-processing filtering block that serves as a benchmark evaluation. The complete evaluation covers results in matrix laboratory (MATLAB) training, validation and testing as well as field-programmable gate array (FPGA) implementation with real-time testing classification.

### 4.2.1 MATLAB Training and Validation Results of CNN Model Without Filter

The benchmark convolutional neural network (CNN) model without filter was trained using the configuration and hyperparameter tuning in Section 3.3.3. Through training in matrix laboratory (MATLAB), the training and validation accuracy plot curves as shown in Figure 4.1 and the training and validation metrics are tabulated and summarized in Table 4.1.

Table 4.1
Training and Validation Metrics for CNN Model Without Filter

| CNN Model | Training Accuracy | Validation Accuracy | Precision | Recall | F1-score | Training Time |
|---|---|---|---|---|---|---|
| Without Filter | **92.43%** | **88.53%** | 92.57% | 92.46% | 92.46% | 8.292 hrs |

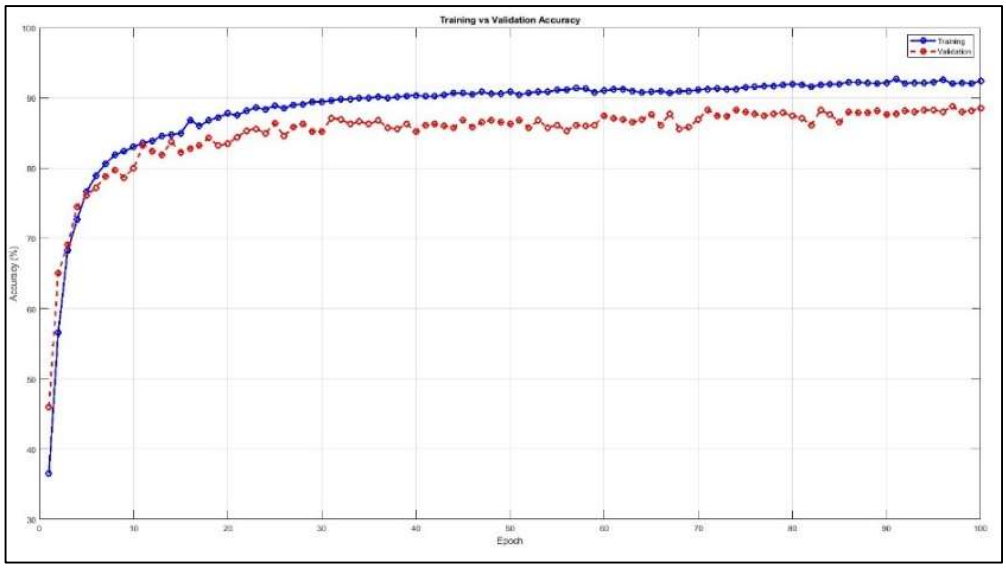



Figure 4.1 Training And Validation Accuracy Plot Of CNN Model Without Filter - Benchmark

Based on Table 4.1, the benchmark CNN model achieved training accuracy and validation accuracy of 92.43% and 88.53% respectively. There's a gap of 3.9% between training and validation which indicates tendency of overfitting, where the CNN performs well but struggles to generalize to the validation dataset. The gap can be noticed in the training and validation accuracy plot in Figure 4.1 that supports the reasoning. Moreover, the validation curves show a little instability compared to the training curve. The precision and recall are, however, closely balanced, scoring 92.57% and 92.46% respectively. This produces a harmonic balance of F1-score of 92.46% that confirms the consistent performance of both metrics. The training time of 8.292 hours presents the computational cost to achieve the metrics performance without applying any filter.

### 4.2.2 MATLAB Testing Results of CNN Model Without Filter

After training, the convolutional neural network (CNN) model was evaluated on unseen testing dataset. Table 4.2 summarizes the testing metrics in matrix laboratory

(MATLAB) and Figure 4.2 displays the confusion matrix that reveals the classification patterns result of the CNN.

Table 4.2
Testing Metrics for CNN Model Without Filter

| CNN Model | Testing Accuracy | Precision | Recall | F1-score | Inference Speed | Testing Time |
|---|---|---|---|---|---|---|
| Without Filter | 69.33% | 72.67% | 69.33% | 70.31% | 0.0578 s | 44.27 s |

Based on Table 4.2, the CNN scores a testing accuracy of 69.33% which is lower than the validation accuracy of 88.53%. This shows a drop of 19.2% of accuracy that indicates the CNN's generalization capability is limited to when exposed to a new dataset. The precision score 72.67% which is higher than the recall score of 69.33%. The difference of precision and recall signifies the CNN makes positive prediction but misclassify some classes. The 2 metrics produces a balanced F1-score of 70.31%. The CNN also scores an inference speed of 0.0578 seconds per sample using Equation (3.15). The testing results of the CNN without filter is further evaluated based on the confusion matrix generated as shown in in Figure 4.2.

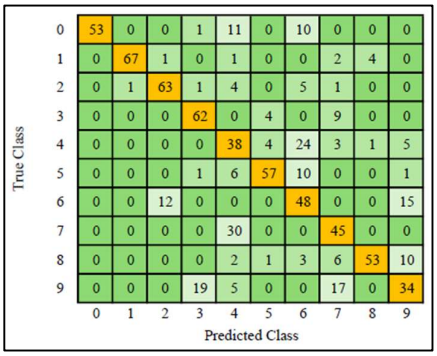



Figure 4.2 Confusion Matrix Of CNN Model Without Filter During MATLAB Testing

The confusion matrix results of the CNN without filter shown in Figure 4.2, shows a moderate diagonal correct prediction as expected that affirms the test accuracy score of 69.33%. However, the model exhibits a significant error in few classes. True class of 4 being the highest misclassification and true class 9 as second class that show major misclassification in this matrix. The total image of the unseen dataset is 750 images which is 15% split of the total dataset. This makes the total images per digit in the test dataset is 75 images. Since true class of 4 and 9 predicted less than half of the total, this marks the 2 lowest classifications among the class. Another obvious

observation is the confusion classification of true class 7 that misclassify as 30 images of class 4, which is almost half the total image despite scoring correct prediction of 45 images.

### 4.2.3 FPGA Implementation Results of CNN Model Without Filter

The field-programmable gate array (FPGA) implementation results of convolutional neural network (CNN) model without filter presents the value of resource utilization and the real-time testing accuracy which is presented in Table 4.3 as well as the confusion matrix from the real-time testing displayed in Figure 4.3. Based on Table 4.3, the CNN utilized 90,612 logic elements (LEs) which takes 79.15% of the Cyclone IV FPGA's total capacity of 114,480 LEs. This shows there are 20.85% balance of LEs available for the additional incorporation of pre-processing block to the system. Besides, the CNN also utilizes 58,900 registers that present the total state storage used for the CNN pipeline.

Table 4.3
FPGA Implementation Results for CNN Model Without Filter

| CNN Model | Total Logic Elements | Total Register | Real-time Testing Accuracy |
|---|---|---|---|
| Without Filter | 90,612 | 58,900 | 14.33% |

The CNN scores real-time testing accuracy of 14.33% which is lower than the matrix laboratory (MATLAB) testing accuracy of 69.33%. This shows a drop of 55% of accuracy that highlights the challenges of hardware implementation as well as the real-time classification evaluation which will be further discussed under Section 4.3.

The confusion matrix results of the CNN without filter during real-time testing shown in Figure 4.3, shows a low classification frequency that proves the low real-time test accuracy score of 14.33%. Figure 4.3 provides insight into the classification pattern of the CNN during FPGA implementation. The prediction in real-time is mostly scattered and the highest prediction scores is 31 images. However, the 31 images misclassify true class **4** as class **5**.

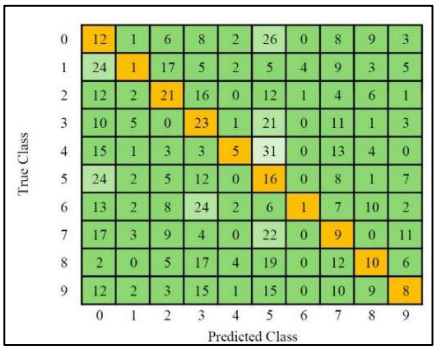



Figure 4.3 Confusion Matrix Of CNN Model Without Filter During Real-Time Testing

### 4.2.4 Summary of the Performance of Benchmark Model

The convolutional neural network (CNN) model without filter succesfully demonstrates the functionality of the 7-layer CNN that was designed for field-programmable gate array (FPGA) hardware implementation. This is shown through its training accuracy score of 92.43% in matrix laboratory (MATLAB) and complete implementation on FPGA with resource utilization of 90,612 logic elements (LEs) and 58,000 registers. Despite the low real-time test accuracy score that reveals the implementation challenges, the CNN established a valid benchmark result for the evaluation on the impact of co-design filter blocks on CNN performance. The complete designed pipeline from MATLAB training through the weight and bias conversion to FPGA deployment was validated. This confirms the feasibility of the FPGA implementation methodology in Section 3.4.

## 4.3 Overall Performance of CNN Models with Filters

This section presents the complete evaluation of four convolutional neural network (CNN) models with the co-design of pre-processing filters which are gaussian, median, bilateral and sobel that has been outlined in Section 3.5. The results of all filter models are presented together for a direct comparison of results representation.

### 4.3.1 MATLAB Training and Validation Results of CNN Models with Filters

The training and validation metrics for all the convolutional neural network (CNN) filter models are tabulated in Table 4.4. The bilateral filter model shows the best performing model that scores the highest training accuracy of 93.97%, which is relatively improved 1.66% over the benchmark result. Bilateral filter model also scores the highest validation accuracy of 92.40%, similar with the median filter model score for validation.

Table 4.4
Training and Validation Metrics of CNN Models with Filters

| CNN Model | Training Accuracy | Validation Accuracy | Precision | Recall | F1-score | Training Time |
|---|---|---|---|---|---|---|
| With Gaussian Filter | 93.33% | 90.27% | 93.65% | 93.66% | 93.64% | 8.098 hrs |
| With Median Filter | 93.09% | 92.40% | 93.13% | 93.12% | 93.09% | 8.214 hrs |
| **With Bilateral Filter** | **93.97%** | **92.40%** | **93.98%** | **93.99%** | **93.97%** | **8.213 hrs** |
| With Sobel Filter | 9.97% | 10.00% | 8.29% | 9.95% | 8.33% | 8.232 hrs |

The validation accuracy of CNN model with bilateral filter also relatively improved 4.37% based on the Equation (3.16) over the benchmark result. Besides, the bilateral filter model scores a nearly perfect balanced precision and recall score of 93.98% and 93.99% respectively, that resulting in a balanced F1-score of 93.97%.

The gaussian filter model scores 93.33% training accuracy and 90.27% validation accuracy, with a training and validation gap of 3.06% that is still smaller than the benchmark gap of 3.9%. This shows a moderate improvement in generalization of the model. On the other hand, the median filter model scores 93.09% training accuracy and 92.40% validation accuracy, resulting in a small training and validation gap of 0.69% which indicates an excellent generalization capability of the model with filters except for CNN with sobel filter.

Looking at sobel filter model, the model fails completely with 9.97% training accuracy and 10% validation accuracy that shows fail performance. The results of sobel filter model are further explained through the training and validation accuracy plot during training in Figure 4.7.

The training time across all models with filters except sobel are range from 8.098 hours to 8.214 hours, which is slightly lower than the benchmark's training hours that took 8.292 hours. This indicates the incorporation of pre-processing block

facilitates faster convergence for the CNN despite the additional pre-processing steps for digit SL classification.

The training and validation results are further elaborated with the plot of training and validation accuracy during training that provides insight into the models' convergence and generalization behaviour. The plots of respective filter models are presented from Figure 4.4 through Figure 4.7.

Looking at the plots of training and validation accuracy of gaussian filter model during training in Figure 4.4, the model present a smoother convergence of the validation accuracy compared to the benchmark CNN model. The gap between training and validation accuracy is smaller that indicates a well generalization and rather have smaller gaps than the gap results of 3.06% obtained at the end of the training.

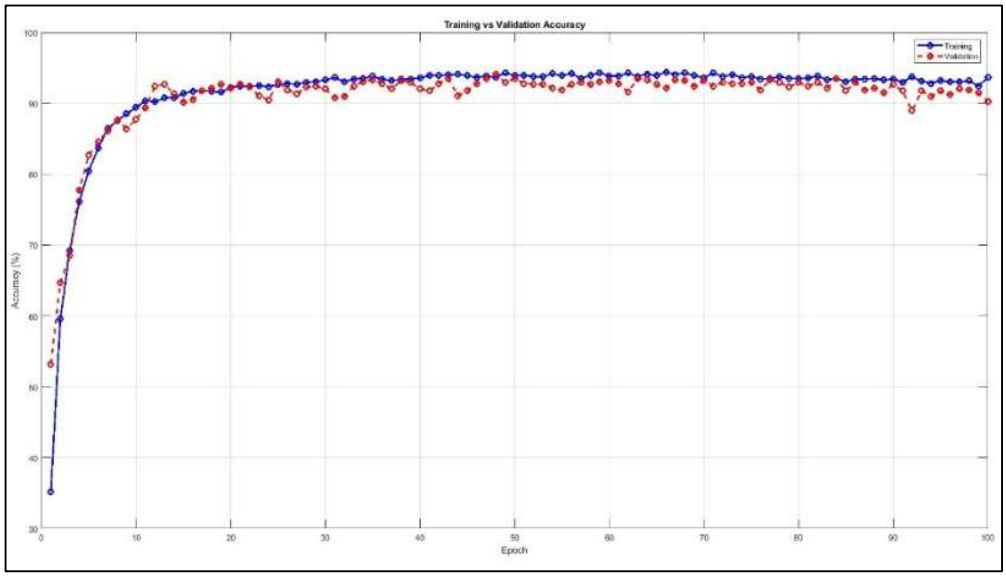



Figure 4.4 Training And Validation Accuracy Plot Of CNN Model With Gaussian Filter

Figure 4.5 displays the training and validation accuracy plot of median filter model. The curve pattern is similar to gaussian filter model, signifying a same smooth convergence. The gap between the training and validation curve rather is less close to one another in comparison to the gaussian filter. The gap is rather slightly off than the final gap of 0.69% but still indicates a well generalization.

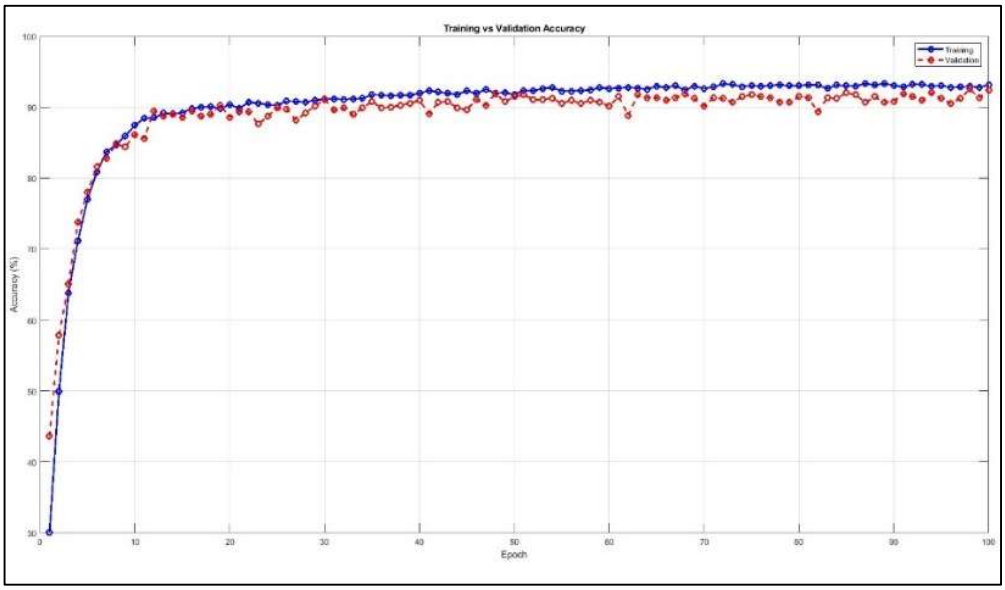



Figure 4.5 Training And Validation Accuracy Plot Of CNN Model With Median Filter

In comparison to Figure 4.4 and Figure 4.5, the accuracy plot of bilateral filter model shows a much better result with the higher accuracy scores during training and small training-validation gap pattern as shown in Figure 4.6. This demonstrates the model effectiveness in digit SL classification with best generalization whereby the training and validation converges closely and achieves the balance of feature preservation and noise reduction.

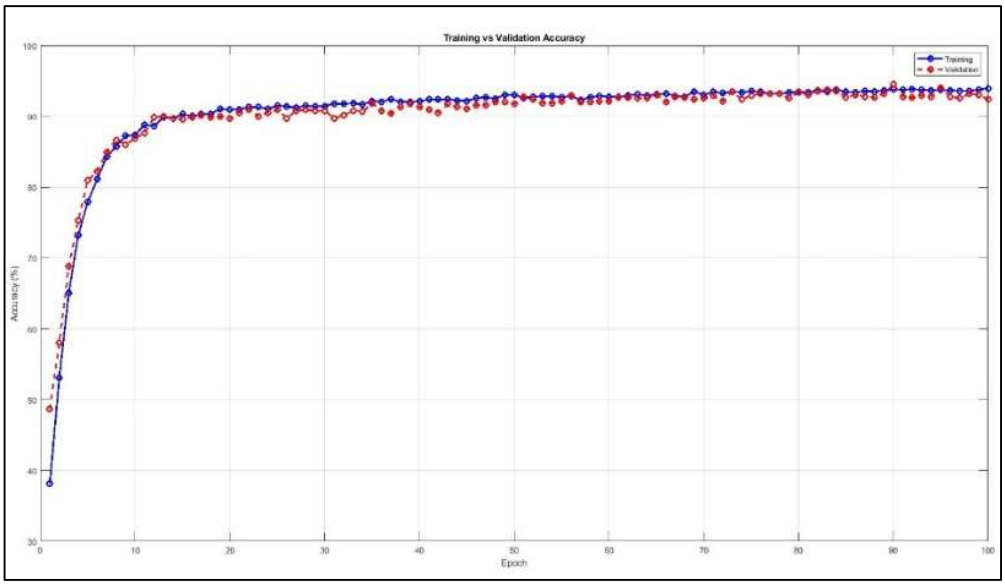



Figure 4.6 Training And Validation Accuracy Plot Of CNN Model With Bilateral Filter

However, the accuracy curve plot of training and validation for sobel filter model proves a failure in training as shown in Figure 4.7. At the early part of the training, the model shows a moderate convergence of curve scoring a range of 70%-75% for training and validation. However, in the later part, the pattern of the validation curve shows instability.

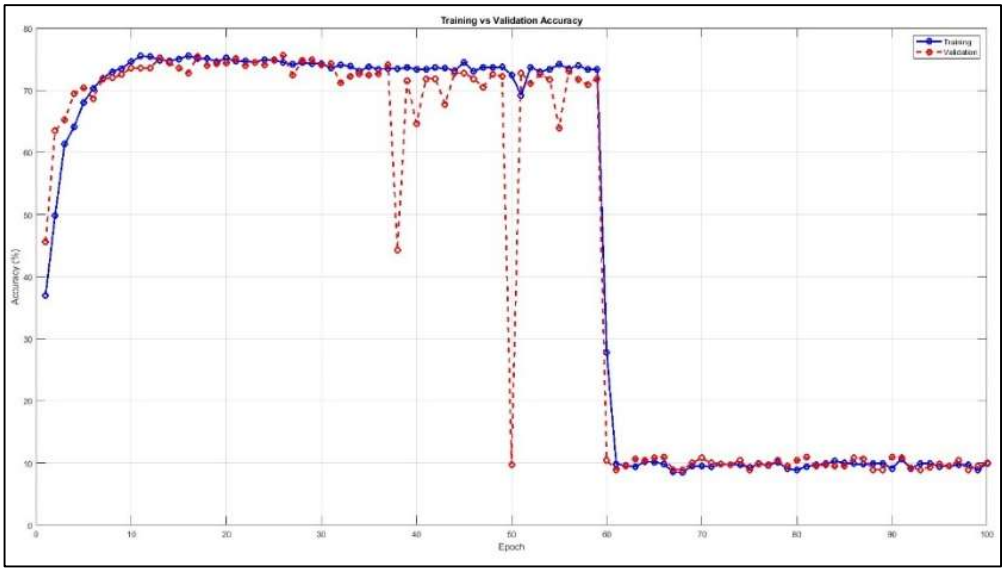



Figure 4.7 Training And Validation Accuracy Plot Of CNN Model With Sobel Filter

There's a drastic drop for both training and validation after the 60$^{th}$ epoch mark that indicates no further learning and a consistent underscore of low accuracy throughout training. The drop primarily due to gradient explosion since sobel filter produces edge maps with sparse and signed gradients values unlike other filters. When the edge maps pass through the CNN, the gradients multiply across each layer during backpropagation. The gradient increases exponentially with depth causing it to become infinitely large. This corrupts all subsequent weights update. The no further learning at the remaining 40 epochs during training affirms the gradient explosion.

### 4.3.2 MATLAB Testing Results of CNN Models with Filters

The testing metrics for all the convolutional neural network (CNN) models with filters are tabulated and summarized in Table 4.5. The metrics in testing consists of the testing accuracy, precision, recall, F1-score, inference speed, testing time and confusion

matrix that provides insight into the classification pattern of the model on unseen testing data set on matrix laboratory (MATLAB) software.

Table 4.5
Testing Metrics of CNN Models with Filters

| CNN Model | Testing Accuracy | Precision | Recall | F1-score | Inference Speed | Testing Time |
|---|---|---|---|---|---|---|
| With Gaussian Filter | 73.60% | 75.72% | 73.60% | 73.37% | 0.0556 s | 42.67 s |
| With Median Filter | 77.20% | 78.18% | 77.20% | 76.97% | 0.0567 s | 43.53 s |
| **With Bilateral Filter** | **77.87%** | **79.63%** | **77.87%** | **77.92%** | **0.0580 s** | **44.20 s** |
| With Sobel Filter | 10.00% | 1.00% | 10.00% | 1.82% | 0.0588 s | 44.51 s |

The results in Table 4.5 shows bilateral filter model again achieves best performing model with 77.87% testing accuracy which is 12.32% relative improvement from the benchmark results. The precision score 79.63% which is higher than then recall score of 77.87%. This indicates that the model is most reliable for positive prediction compared to other models. The F1-score confirms the balanced improvement of both precision and recall by scoring 77.92%. The median filter model achieves closely with the bilateral filer model with 77.20% testing accuracy, which is 11.35% relative improvement from the benchmark result. The model also scores a precision of 78.18% with recall of 77.20% showing excellent balance of performance. The gaussian filter model also improve its testing accuracy of 73.60% from the benchmark but lower than the bilateral and median filter model. Similarly, the precision and recall score is balance as well but with lower score percentage. However, the sobel filter model confirms its complete failure by scoring 10% of testing accuracy. With its precision drop of 1% shows no correct positive predictions by the model. The inference speed of all CNN model with filters scores a consistent range from 0.0556 to 0.0588 seconds per sample. This demonstrates the incorporation of pre-processing filter does not compromise the computation efficiency during inference. The testing evaluation is further supported with the result of confusion matrix generated in MATLAB that provide insights to the classification pattern of each CNN models. The confusion matrix is presented in Figure 4.8 through Figure 4.11 confusion matrix obtained in testing for each CNN model provides a specific understanding of the classification and misclassification patterns.

The confusion matrix of gaussian filter model shown in Figure 4.8 displays an improved diagonal correct predictions compared to the benchmark results that improve in testing accuracy from 69.33% to 73.60%. Most class scores above half the total image

per class, except for true class **9** that shows the lowest classification by scoring only 25 correct images and most misclassified it as class **8** highest and **3** second highest.

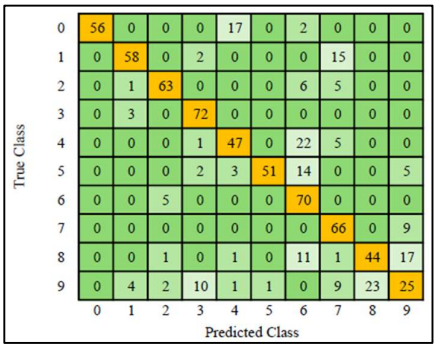



Figure 4.8 Confusion Matrix Of CNN Model With Gaussian Filter For MATLAB Testing

A notable similar pattern to Figure 4.2 is that the true class 0 score a high correct prediction and misclassify other prediction mostly to class 4. Same goes to true class 1 mostly misclassify as class 7, true class 4 mostly misclassify as class 6, true class 5 mostly misclassify as class 6 and true class 8 mostly misclassify as class 9.

The confusion matrix of median filter model shown in Figure 4.9 displays a more improved diagonal correct predictions compared to the gaussian filter model and benchmark results. This proves the testing accuracy score of 77.20%, which is higher than the gaussian filter model and benchmark results.

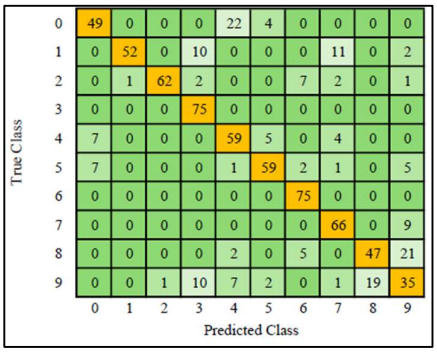



Figure 4.9 Confusion Matrix of CNN model with median filter for MATLAB testing

The rest of the class shows slight improvement, and the most obvious is true class **9** predicts 35 images correctly, which is 10 images more than the gaussian filter model. At the same time, the pattern of the CNN misclassification of true class **9** as

class **3** and **8** still persists. Similar notable pattern with Figure 4.2Figure 4.7 and Figure 4.8 is that the true class **0** score a high classification but shows similar misclassification pattern to class **4.** Same goes to true class **1** mostly misclassify as class **7**, and true class **8** mostly misclassify as class **9**.

The confusion matrix of bilateral filter model shown in Figure 4.10 display the best diagonal correct predictions compared to all other models where most classification of true class is improved. This futher proves the highest testing accuracy of 77.87% score among the models.

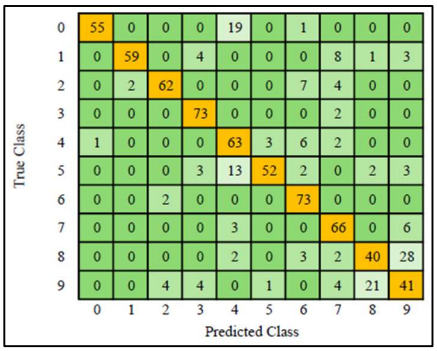



Figure 4.10 Confusion Matrix Of CNN Model With Bilateral Filter For MATLAB Testing

The classification of true class **9** past the half of total image mark, with 41 images correct prediction. The pattern of Figure 4.2, Figure 4.8 and Figure 4.9 is the same where the ture class **0** score a high correct prediction and shows similar pattern of misclassification to class **4.** True class **5** mostly misclassify as class **4**, true class **8** misclassify as class **9** and true class **9** misclassify as class **8** while other patterns has improves.

Finally, the confusion matrix of sobel filter model shown in Figure 4.11, displays the worst-case scenario for classification, whereby all class is classified to class 3. True class 3 will be an exception as a correct prediction due to the consistent misclassification of other classes which produces the 10% testing accuracy obtained.

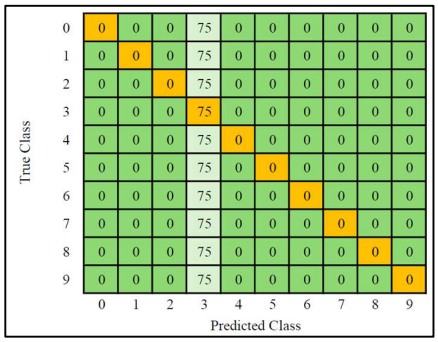



Figure 4.11 Confusion Matrix Of CNN Model With Sobel Filter For MATLAB Testing

### 4.3.3 FPGA Implementation Results of CNN Models with Filters

In this section, all convolutional neural network (CNN) models with filters were implemented on the field-programmable gate array (FPGA) and undergoes the same process as the benchmark model without filter. Upon completion of the FPGA implementation, the resource utilization information is obtained from the compilation report in Quartus, and the real-time testing accuracy is conducted on the same unseen testing dataset in matrix laboratory (MATLAB) which is displayed via mobile phone. The results of the FPGA resource utilization and real-time performance of CNN models with filters are tabulated in Table 4.6. The real-time testing accuracy result is further explained based on the confusion matrix presented in Figure 4.12 through Figure 4.15, that provides insights into the classification pattern of the models in FPGA implementation during real-time.

Table 4.6
FPGA Resource Utilization and Real-time Performance of CNN Models with Filters

| CNN Model | Total Logic Elements | Total Register | Real-time Testing Accuracy |
|---|---|---|---|
| With Gaussian Filter | 109,582 | 69,227 | 15.60% |
| With Median Filter | 109,575 | 69,275 | 16.93% |
| **With Bilateral Filter** | **110,582** | **69,245** | **17.33%** |
| With Sobel Filter | 109,611 | 29,226 | 13.47% |

Based on Table 4.6, CNN model with bilateral filter utilizes most logic elements (LEs) of 110,582 with 22.04% relative increase from benchmark and achieves highest real-time testing accuracy of 17.33%. This increased number of resource utilization across all models compared to the benchmark justified the real-time accuracy where the

additional resource contributes to the improvement of performance except for sobel filter model. The median filter model on the other hand, scores 16.93% real-time testing accuracy which offers a comparable performance while utilize fewer resources of 109,575 LEs. This presents an alternative to the bilateral filter model due to its real-time testing accuracy higher than the benchmark result. The gaussian filter model achieves 15.6% real-time testing accuracy with 109,582 utilized LEs shows slightly higher accuracy than the benchmark score.

The number of registers utilized across all the models except sobel filter model, range from 69,227 to 69,275 registers signifies increase of registers used over the benchmark model that utilized 58,900 registers. The increase is due to the additional pipeline for the filter application include the 3x3 window line buffer formation, multi-stage sorting network design for median filter and sequential multiplication logic in gaussian filter design. The registers are used to store the pixel values and controls the signals as the data flows through pre-processing filter block before reaching the CNN block.

The sobel filter model, however, shows an interesting result of utilizing 109,611 LEs to the other models but register usage of only 29,266, which is near half in total compared to other models. The small number of registers reflects the software training failure where available data paths are not required in the model. Without meaningful weights from trained CNN model in MATLAB, the resource utilization in FPGA is minimal and contributes to the poor performance. At the same time, the real-time testing accuracy of 13.47% is lower than the benchmark results affirm the model performance failure. Similar to the benchmark model, real-time testing accuracy is and proceed on the same unseen testing data displayed via mobile phone and evaluated in the form of confusion matrix which is presented in Figure 4.12 through Figure 4.15.

The confusion matrix of gaussian filter model is as shown in Figure 4.12. The diagonal part of the confusion matrix is coloured in yellow as an ideal classificaiton mark to see the total image classified correctly for every class despite the low classification results of the model. The confusion matrix proves the low accuracy of 15.60% obtained by the model for real-time testing.

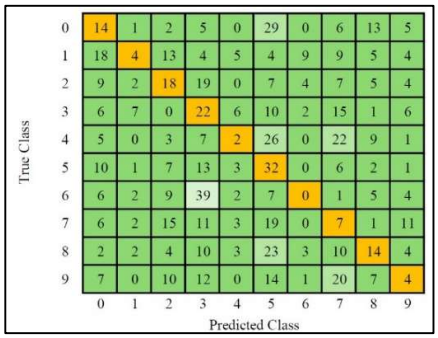



Figure 4.12 Confusion Matrix Of CNN Model With Gaussian Filter For FPGA Testing

The model scores the highest classification for true class **5** with 32 correct prediction and second is true class **3** with 22 images. The misclassification can been seen in most class where true class **0** mostly misclassified as class **5**, true class **4** mostly misclassified as 26 image of class **5** and 22 image of class **7**, true class **8** mostly misclassified as class **5,** and true class **9** mostly misclassified as class **7**. The highest miclassifcation is true class **6** which is mostly misclassified as 39 image of class **3**. In sum, the pattern of the matrix shows the model misclassify a total of 3 true class (true class **0**,**4** and **8**) as class **5.**

The confusion matrix of median filter model is as shown in Figure 4.13. Similarly, the confusion matrix proves the low accuracy of 16.93% obtained by the model for real-time testing but improved from the bencmark model and gaussian filter model. The model scores the highest classification for class **3** with 26 correct predictions.

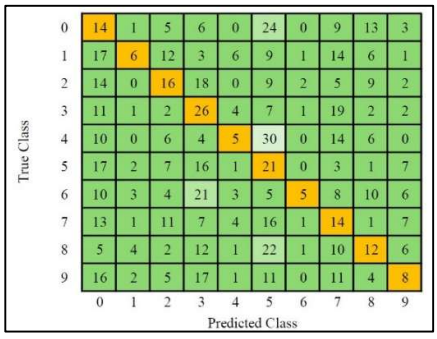



Figure 4.13 Confusion Matrix Of CNN Model With Median Filter For FPGA Testing

The misclassification in Figure 4.13 is almost similar to Figure 4.12 where true class **0** mostly misclassified as class **5**. Then, true class **4** with the highest

misclassification of 30 image of class **5.** True class **6** mostly misclassified as class **3,** and finally, true class **8** mostly misclassified as class **5**. The pattern of the matrix also shows the model misclassify the same 3 true class as class **5** as in Figure 4.12**.**

The confusion matrix of bilateral filter model is as shown in Figure 4.14. Similarly, the confusion matrix proves the highest score accuracy of 17.33% obtained by the model in comparison to other model.

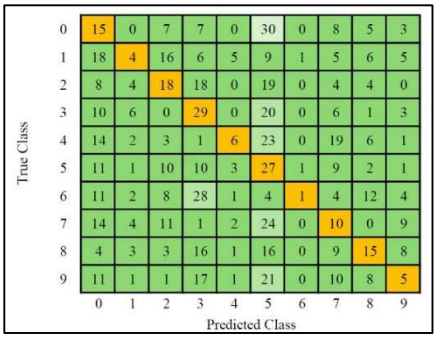



Figure 4.14 Confusion Matrix Of CNN Model With Bilateral Filter For FPGA Testing

The model scores the highest classification for true class **3** with 29 correct images, following with true class **5** with 27 correct images. However, true class 5 high number of classification is debatable due to its impact of misclassification on other true class. The misclassification occurs on true class **0** that mostly misclassified as 30 image of class **5,** which scores the highest misclassification. The other 4 true class which are true class **3**, **4**, **7** and **9** which are mostly classified as class 5 with 20, 23, 24 and 21 image respectively. Another notable result is true class **6** which mostly misclassified as class **3**, true class **1** mostly misclassify as class **0**. True class **2** have equal classification and misclassification with class **3**. Similarly, true class **8** have almost equal classification and misclassification with class **3** and **5**. The similar results shows the confusion of prediciton by the CNN.

Finally, the confusion matrix of sobel filter model in Figure 4.15. The matrix of the model suprising shows somes correct classification in real-time. The model scores the highest classification of true class **5** with 21 correct image.

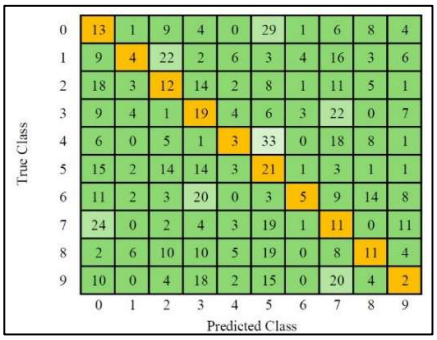



Figure 4.15 Confusion Matrix Of CNN Model With Sobel Filter For FPGA Testing

Meanwhile, the other class shows a scattering misclassification where true class **0** mostly misclassified as class **5**, true class **1** mostly misclassified as class **2**, true class **3** mostly misclassified as class 7, true class **6** mostly misclassified as class **3**, true class **7** mostly misclassified as class **0** and true class **9** mostly misclassified as class **7**. True class **4** scores the highest misclassification of 33 image class of **5**. The matrix results prove the accuracy score of 13.47% in real-time testing and still lower than the benchmark result of 14.33%.

### 4.3.4 Summary of the Performance of CNN Models with Filters

The overall evaluation of CNN models with filters shows three filters which are gaussian, median and bilateral has improved the CNN performance consistently across all metrics compared to the benchmark. The bilateral filter model achieved the best overall results among the filter models in training accuracy of 93.97%, validation accuracy of 92.40%, testing accuracy of 77.87% and real-time testing accuracy of 17.33%. Median filter model came in second performed model with well generalization and score a small training and validation gap of 0.69%. While gaussian achieve a moderate performance but with improvement over the benchmark. The sobel filter model, however, shows complete failure and proved unsuitable for digit sign language (SL) classification. In overall of the success filter models, the resource utilization during hardware implementation relatively increased by approximately 21% which is still under the capacity of the FPGA's resource. At the same time, the training times remained consistent for all filter models except for sobel between 8.1 and 8.3 hours that indicates the incorporation of co-design filter does not introduce any computational overhead during training.

## 4.4 Comparative Analysis and Discussion

In this section, a comprehensive comparative analysis is conducted between the benchmark convolutional neural network (CNN) model and the applied filter model. This section examines the impact and improvements of filter block to CNN, the hardware-software discrepancies and the key findings.

### 4.4.1 Comparison of Overall Performance of all CNN Models

The complete overall performance of all five convolutional neural network (CNN) models across all evaluation stages is tabulated in Table 4.7. Bilateral filter model outperforms all other models across all evaluation that achieves the highest training accuracy of 93.97%, validation accuracy of 92.40%, testing accuracy of 77.87% and real-time testing accuracy of 17.33%.

Table 4.7
Overall Performance Summary of all CNN Models

| CNN Model | Training Accuracy | Validation Accuracy | Testing Accuracy | Real-time Testing Accuracy | Total Logic Elements | Total Register |
|---|---|---|---|---|---|---|
| Without Filter | 92.43% | 88.53% | 69.33% | 14.33% | 90,612 | 58,900 |
| With Gaussian Filter | 93.33% | 90.27% | 73.60% | 15.60% | 109,582 | 69,227 |
| With Median Filter | 93.09% | 92.40% | 77.20% | 16.93% | 109,575 | 69,275 |
| **With Bilateral Filter** | **93.97%** | **92.40%** | **77.87%** | **17.33%** | **110,582** | **69,245** |
| With Sobel Filter | 9.97% | 10.00% | 10.00% | 13.47% | 109,611 | 29,226 |

Besides, the median filter model scores the same validation accuracy of 92.40% that resulting of the smallest gap between training and validation of 0.69% difference, indicating the well generalization of the model. The gaussian filter model on the other hand, shows a moderate improvement across metric over the benchmark model results. The sobel filter model, however, shows a complete failure with the score training accuracy of 9.97% and testing accuracy of 10%. The results of accuracy metrics show a random guessing for a 10-class digit sign language classification through the pattern of confusion matrix result in MATLAB. The failure is due to the filter producing edge maps that discard the grayscale intensity, which lose the critical information for feature extraction in convolutional neural network (CNN).

The comparative analysis is further discussed based on the relative performance improvement of the CNN models with filters over the benchmark CNN model without

filter. The relative performance improvement across all evaluations is tabulated and elaborated comprehensively in Table 4.8

Table 4.8
Relative Performance Improvements over Benchmark Results Across all Evaluation

| CNN Model | Training Accuracy | Validation Accuracy | Testing Accuracy | Real-time Testing Accuracy |
|---|---|---|---|---|
| Without Filter | 92.43% | 88.53% | 69.33% | 14.33% |
| With Gaussian Filter | 93.33% (+0.97%) | 90.27% (+1.97%) | 73.60% (+6.16%) | 15.60% (+8.86%) |
| With Median Filter | 93.09% (+0.71%) | 92.40% (+4.37%) | 77.20% (+11.35%) | 16.93% (+18.14%) |
| **With Bilateral Filter** | **93.97% (+1.67%)** | **92.40% (+4.37%)** | **77.87% (+12.32%)** | **17.33% (+20.94%)** |
| With Sobel Filter | 9.97% (-89.21%) | 10.00% (-88.7%) | 10.00% (-85.58%) | 13.47% (-6%) |

Based on Table 4.8, the relative improvement hierarchy is consistent across all evaluation. Bilateral filter model achieves the highest relative improvement of +1.67% in training accuracy, +4.37% in validation accuracy, +12.32% in matrix laboratory (MATLAB) testing accuracy and +20.94% in real-time testing accuracy. Median filter model comes second in the hierarchy achieving relative performance of +0.71% in training accuracy, +4.37% in validation accuracy, +11.35% in MATLAB testing accuracy and +18.14% in real-time testing accuracy. Then, gaussian filter model comes in third with achieving relative performance of +0.97% in training accuracy, +1.97% in validation accuracy, +6.16% in MATLAB testing accuracy and +8.86% in real-time testing accuracy. While sobel filter model performance failed for digit SL classification.

The consistent improvement across all evaluation by the 3 model with gaussian, median and bilateral filter, proves the feasibility of enhancing CNN performance using the co-design approach of pre-processing block. Critically, the improvements observed in MATLAB translates an even larger relative improvements in field-programmable gate array (FPGA) implementation, with bilateral filter model achieve +20.94% improvement in real-time testing compared to +12.32% improvement in MATLAB testing despite the low accuracy value. The improvement in FPGA indicates that importance of co-design filtering block when dealing with the challenges of real-time implementation, include the quantization effects and environmental variations. The preservation of performance hierarchy across MATLAB and FPGA validates the success of co-design approach for the filter block design in CNN.

### 4.4.2 Impact of Co-design Filter on CNN performance

The results in Section 4.4.1 have demonstrated that the co-designed filters improve the convolutional neural network (CNN) performance across all evaluation significantly with bilateral filter model achieve as the best relative performing model. While median filter in second and gaussian filter in third of the hierarchy performance. Sobel filter, however, completely failed. The improvement validates the co-design approach where filters are first design in hardware to consider the hardware constraints and then adapt the design in matrix laboratory (MATLAB) software. The preserved hierarchy performance shows that the filter's behaviour is maintained effectively from MATLAB to field-programmable gate array (FPGA) platform. Edge-preserving filter like bilateral and median outperforms the gaussian filter that conduct simple smoothing due to the maintaining of shape boundaries while reducing noise. This aligns with the findings in of filters preserving critical features for disease detection in medical imaging, which proven the effectiveness of the edge-preserving filter [89], [92], [93]. Furthermore, the co-design approach maintains computational efficiency with consistent training times which is shorter than the benchmark model. This indicates no overhead from the incorporation of filter. Moreover, the increased of resource utilization by approximately 21% for the incorporation of filter shows a practical implementation with the consideration of FPGA hardware constraints that motivates the realization of application-specific integrated circuit (ASIC) application [8], [15].

### 4.4.3 Hardware-Software Discrepancy Performance

One critical finding of this research is the performance gap of convolutional neural network (CNN) models in MATLAB and field-programmable gate array (FPGA) platform, ranging from 55% to 60% of absolute differences for the successful CNN model in digit sign language (SL) classification. The gap validates the fundamental challenge of system achieving high performance at software level but overlook the hardware constraints.

There are 3 main factors of the discrepancy where the first is due to the quantization effect in converting the 32-bit floating-point weights to a 16-bit fixed-point representation which fits the FPGA's compatibility. The quantization introduces cumulative rounding errors during the conversion. Alternative strategies for

quantization challenges exist that can overcome the rounding errors. One of the strategies is using binarized neural network (BNN) which able to reduce wights to 1-bit and lowers the memory footprint, but it suffers accuracy loss for SL classification [17]. Another strategy is using a dynamic 8-bit quantization with layer-wise precision that can adapt fractional bits per layer to minimize quantization error while maintaining efficiency in real-time [8], [48]. However, this requires complete retraining or more logic elements utilization. The 16-bit fixed point approach was chosen to balance the hardware simplicity and accuracy preservation on the DE2-115 FPGA.

Next, the timing constraints in FPGA introduces latency that affects the frame-to-frame consistency. Finally, the environmental variations during real-time testing. Although the testing conduction in a dark room with max brightness of the mobile phone, the brightness conditions and camera blur may affect the data acquisition of the camera.

Despite the absolute gap in testing accuracy, the relative improvement of the filters persists. The improvement in performance of CNN is due to the noise reduction and edge preserving capabilities of filters becomes more important when dealing with hardware-induced noise from quantization and environmental factors. The preserved improvement hierarchy of filter performance throughout all evaluation affirms the co-design approach successfully capture the relative differences, despite the compromise of testing accuracy. This shows that the co-design filters are feasible for real-time FPGA implementation in realizing the application-specific integrated circuit (ASIC) applications.

## 4.5 Summary

This chapter has evaluated the five convolutional neural network (CNN) models, with and without filter, for digit sign language classification from matrix laboratory (MATLAB) training to field-programmable gate array (FPGA) implementation. The CNN model without filter has achieved 92.43% training accuracy, 88.53% validation accuracy and 69.33% testing accuracy in MATLAB and was implemented in FPGA which utilized 90,612 logic elements (LEs) and scores 14.33% real-time testing accuracy. The results are served as a benchmark for the comparison with CNN model with applied co-design filters.

Three filters which are gaussian, median and bilateral, has successfully improved the performance of CNN model. Bilateral filter model achieved the best results of training accuracy of 93.97%, validation accuracy of 92.40%, testing accuracy of 77.87% and real-time testing accuracy of 17.33%, where all are relatively improved over the benchmark results. The median filter model achieved the second-best results of training accuracy of 93.09%, validation accuracy of 92.40%, testing accuracy of 77.20% and real-time testing accuracy of 16.93%, where all are also relatively improved over the benchmark results. Next, the gaussian filter model comes in third by achieving training accuracy of 93.33%, validation accuracy of 90.27%, testing accuracy of 73.60% and real-time testing accuracy of 15.60%, where the results too are relatively improved. Finally, the sobel filter model has failed completely.

The consistent performance hierarchy from MATLAB training to FPGA implementation validates the co-design approach of design the filtering block. Despite the accuracy gap between MATLAB and real-time testing due to quantization, timing constraints and environmental factors, the relative improvement is amplified in FPGA. This affirms the importance of co-design filter in real-time challenges. In sum, CNN-based digit SL classification on FPGA using a co-design approach is feasible.

# CHAPTER 5
# CONCLUSION

## 5.1 Introduction

This research addresses the issue of sign language (SL) recognition and classification by using convolutional neural network (CNN) classification model deployed on field-programmable gate array (FPGA) for real-time applications. Many approaches for SL classification have been utilizing deep learning for accurate classification, yet the real-time application still uses hardware which is bulky and inconvenient for users. Moreover, available portable hardware for SL classification underperforms due to CNN computational demands for real-time deployment.

The first objective of this study was to design and develop a functional digit SL classification model using convolutional neural network (CNN) such that the architecture can be implemented on hardware. This objective was achieved when the developed 7-layer CNN model scored 92.43% in training accuracy, 88.53% in validation accuracy, 69.33% testing accuracy in matrix laboratory (MATLAB) software and was successfully implemented on FPGA using 90,612 logic elements (LEs) with 79% left of available resources as presented in Section 4.2.

The second objective was to enhance CNN performance for digit SL classification using the hardware-software co-design approach of incorporating pre-processing filter block. The second objective is also achieved by implementing the four filters which are gaussian, median, bilateral and sobel, first in the FPGA in consideration of the hardware constraints and pipeline. Then, adapt and mirror the behavior of the design in MATLAB for training. Bilateral filter model achieved the best relative improvements of +12.32% in MATLAB testing accuracy and +20.94% in real-time testing accuracy on FPGA over the benchmark results. The median filter model comes in second of the performance hierarchy with +11.35 in MATLAB testing accuracy and +18.14% in real-time testing accuracy, while gaussian filter model scores performance of +6.16% in MATLAB testing accuracy and +8.86% in real-time testing accuracy, as presented in in Section 4.3. Sobel filter however shows a complete failure throughout the evaluation which requires further study in future work.

Finally, the third objective is to evaluate the feasibility of CNN model with co-design filter block for a real-time digit SL classification that is implemented in FPGA. The evaluation in Section 4.4 shows the achievement of this objective where the feasibility is confirmed through the persistent and consistent performance hierarchy across all platforms. The filter's benefits were amplified in FPGA through the relative improvement results, and all models were designed within the FPGA resource limitation. All filter models successfully implemented within the Cyclone IV Terasic DE2-115 FPGA development board's total logic element capacity, utilizing 95% to 97% of available resources. This demonstrates the practical feasibility of the whole designed system within the hardware constraints.

The limitations of the system prototype for SL classification are addressed in this study. The FPGA and camera used for this study may influence the performance of the system in comparison to a higher performance FPGA. However, the FPGA is feasible to perform for CNN hardware implementation and its performance in digit SL classification will prove the potential of ASIC application in future works.

In summary, the research contributes to a success enhanced system FPGA prototype for SL classification by incorporating co-design filter within the CNN model that is optimized with hardware constraints. The results demonstrate that the bilateral filter improved CNN performance in both software and hardware for digit SL classification. Ultimately, the study demonstrates a potential improved solution that addresses the challenges and limitations that benefit society, especially the deaf or hard of hearing (DHH) community.

## 5.2 Research Contributions

The significance of this study lies in the incorporated co-design image filtering block that impacts the final performance for sign language (SL) classification. By modifying the convolutional neural network (CNN) model with suitable co-design image filtering, the study aims to enhance the overall performance of the final prototype that can lead to development of application-specific integrated circuits (ASICS) via the success implementation of field-programmable gate array (FPGA). The results show that the edge-preserving filter, bilateral filter in particular, improve the performance of CNN in both software and hardware, validating the co-design approach.

The advantages of CNN model for digit SL classification include its ability to learn complex features automatically which benefits the overall system. The CNN capabilities are highlighted in this study as it is crucial for classifying SL with high accuracy, especially in real-time. Several evaluations were conducted in this research to analyze the performance of the CNN model in SL classifications. The analysis consists of important metrics such as training, validation and testing accuracy, precision, recall and F1-score which represents the overall performance that reveals the potential improvement for the CNN models.

The main contribution of this research is the hardware-software co-design approach of incorporating pre-processing filtering block first implemented in FPGA pipeline then mirrored in matrix laboratory (MATLAB) for training. This approach is proved effective based on the consistent hierarchy performance across both platforms. This affirms by the filter's behavior and fundamental functions are similar in MATLAB and FPGA with relative improvements increasing from 6% to 12% in MATLAB to 9% to 21% in FPGA implementation despite the absolute difference in accuracy. The amplification of performance signifies the importance of co-design filters under real-time challenges. The co-design approach validates the framework of developing CNN-based systems where consideration of hardware constraints guides the software design at the early stages of development.

## 5.3 Future Works

For future work, the findings suggest further exploration of sophisticated quantization methods like using binarized neural networks (BNN) and dynamic 8-bit quantization to bridge the gap between software simulations and hardware deployment, particularly involving field-programmable gate array (FPGA) platform. An awareness of developing the system based on the hardware constraints becomes the main consideration to overcome this challenge. In addition, the hardware-software discrepancy demands more insight testing methodologies to bridge the gap of simulation and hardware in the development process. Moreover, a larger dataset would give a fair data split ratio that allows comprehensive evaluation overall and a possible expansion of the variability of the dataset for a real-world simulation evaluation.

# REFERENCES


[1] J. Shin, A. S. Musa Miah, M. A. M. Hasan, K. Hirooka, K. Suzuki, H. S. Lee, and S. W. Jang, 'Korean Sign Language Recognition Using Transformer-Based Deep Neural Network', *Applied Sciences (Switzerland)*, vol. 13, no. 5, Mar. 2023, doi: 10.3390/app13053029.

[2] X. Han, F. Lu, J. Yin, G. Tian, and J. Liu, 'Sign Language Recognition Based on R(2+1)D with Spatial-Temporal-Channel Attention', *IEEE Trans. Hum. Mach. Syst.*, vol. 52, no. 4, pp. 687–698, Aug. 2022, doi: 10.1109/THMS.2022.3144000.

[3] G. Z. de Castro, R. R. Guerra, and F. G. Guimarães, 'Automatic translation of sign language with multi-stream 3D CNN and generation of artificial depth maps', *Expert Syst. Appl.*, vol. 215, Apr. 2023, doi: 10.1016/j.eswa.2022.119394.

[4] M. M. Balaha, S. El-Kady, H. M. Balaha, M. Salama, E. Emad, M. Hassan, and M. M. Saafan, 'A vision-based deep learning approach for independent-users Arabic sign language interpretation', *Multimed. Tools Appl.*, vol. 82, no. 5, pp. 6807–6826, Feb. 2023, doi: 10.1007/s11042-022-13423-9.

[5] B. Svendsen and S. Kadry, 'Comparative Analysis of Image Classification Models for Norwegian Sign Language Recognition', *Technologies (Basel).*, vol. 11, no. 4, Aug. 2023, doi: 10.3390/technologies11040099.

[6] D. T. D. M. Dahanayaka, B. G. D. A. Madhusanka, and I. U. Atthanayake, 'A Multi-Modular Approach for Sign Language and Speech Recognition for Deaf-Mute People', *Engineer: Journal of the Institution of Engineers, Sri Lanka*, vol. 54, no. 4, p. 97, Dec. 2021, doi: 10.4038/engineer.v54i4.7474.

[7] S. Katoch, V. Singh, and U. S. Tiwary, 'Indian Sign Language recognition system using SURF with SVM and CNN', *Array*, vol. 14, Jul. 2022, doi: 10.1016/j.array.2022.100141.

[8] C. C. Wang, Y. C. DIng, C. Te Chiu, C. T. Huang, Y. Y. Cheng, S. Y. Sun, C. H. Cheng, and H. K. Kuo, 'Real-Time Block-Based Embedded CNN for Gesture Classification on an FPGA', *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 10, pp. 4182–4193, Oct. 2021, doi: 10.1109/TCSI.2021.3100109.

[9] K. Tamizhelakkiya, S. Gauni, and P. Chandhar, 'Modulation classification

analysis of CNN model for wireless communication systems', *AIMS Electronics and Electrical Engineering*, vol. 7, no. 4, pp. 337–353, 2023, doi: 10.3934/electreng.2023018.

[10] V. Rawal, P. Prajapati, and A. Darji, 'Hardware implementation of 1D-CNN architecture for ECG arrhythmia classification', *Biomed. Signal Process. Control*, vol. 85, Aug. 2023, doi: 10.1016/j.bspc.2023.104865.

[11] C. Te Chiu, Y. C. Ding, W. C. Lin, W. J. Chen, S. Y. Wu, C. T. Huang, C. Y. Lin, C. Yu Chang, M. J. Lee, S. Tatsunori, T. Chen, F. Yi Lin, Y. H. Huang, 'Chaos LiDAR Based RGB-D Face Classification System With Embedded CNN Accelerator on FPGAs', *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 12, pp. 4847–4859, Dec. 2022, doi: 10.1109/TCSI.2022.3190430.

[12] N. M. Alharthi and S. M. Alzahrani, 'Vision Transformers and Transfer Learning Approaches for Arabic Sign Language Recognition', *Applied Sciences (Switzerland)*, vol. 13, no. 21, Nov. 2023, doi: 10.3390/app132111625.

[13] K. A. Shams, M. R. Reaz, M. R. U. Rafi, S. Islam, M. S. Rahman, R. Rahman, M. T. Reza, M. Z. Parvez, S. Chakraborty, B. Pradhan, A. Alamri, 'MultiModal Ensemble Approach Leveraging Spatial, Skeletal, and Edge Features for Enhanced Bangla Sign Language Recognition', *IEEE Access*, vol. 12, pp. 83638–83657, 2024, doi: 10.1109/ACCESS.2024.3410837.

[14] J. P. Sahoo, A. J. Prakash, P. Pławiak, and S. Samantray, 'Real-Time Hand Gesture Recognition Using Fine-Tuned Convolutional Neural Network', *Sensors*, vol. 22, no. 3, Feb. 2022, doi: 10.3390/s22030706.

[15] F. Benevenuti, A. B. De Oliveira, I. C. Lopes, F. L. Kastensmidt, N. Added, V. A. P. Aguiar, N. H. Medina, and M. A. Guazzelli, 'Heavy Ions Testing of an All-Convolutional Neural Network for Image Classification Evolved by Genetic Algorithms and Implemented on SRAM-Based FPGA', in *2019 19th European Conference on Radiation and Its Effects on Components and Systems, RADECS 2019*, 2022, pp. 141–144. doi: 10.1109/RADECS47380.2019.9745650.

[16] I. P. Sari, 'Closer Look at Image Classification for Indonesian Sign Language with Few-Shot Learning Using Matching Network Approach', *International Journal on Informatics Visualization*, vol. 7(3), pp. 638–643, Sep. 2023, doi:

http://dx.doi.org/10.30630/joiv.7.3.1320.

[17] M. Jaiswal, V. Sharma, A. Sharma, S. Saini, and R. Tomar, 'Quantized CNN-based efficient hardware architecture for real-time hand gesture recognition', *Microelectronics J.*, vol. 151, Sep. 2024, doi: 10.1016/j.mejo.2024.106345.

[18] M. Rivera-Acosta, S. Ortega-Cisneros, J. Rivera, and F. Sandoval-Ibarra, 'American sign language alphabet recognition using a neuromorphic sensor and an artificial neural network', *Sensors (Switzerland)*, vol. 17, no. 10, Oct. 2017, doi: 10.3390/s17102176.

[19] S. M. Rayeed, 'American Digit Sign Language Dataset', *Kaggle*, 2021. https://www.kaggle.com/datasets/rayeed045/american-sign-language-digit-dataset (accessed Apr. 17, 2025).

[20] A. S. M. Miah, J. Shin, M. A. M. Hasan, and M. A. Rahim, 'BenSignNet: Bengali Sign Language Alphabet Recognition Using Concatenated Segmentation and Convolutional Neural Network', *Applied Sciences (Switzerland)*, vol. 12, no. 8, Apr. 2022, doi: 10.3390/app12083933.

[21] S. Sharma and S. Singh, 'Vision-based hand gesture recognition using deep learning for the interpretation of sign language', *Expert Syst. Appl.*, vol. 182, Nov. 2021, doi: 10.1016/j.eswa.2021.115657.

[22] V. Jain, A. Jain, A. Chauhan, S. S. Kotla, and A. Gautam, 'American Sign Language recognition using Support Vector Machine and Convolutional Neural Network', *International Journal of Information Technology (Singapore)*, vol. 13, no. 3, pp. 1193–1200, Jun. 2021, doi: 10.1007/s41870-021-00617-x.

[23] Y. Gu, Sherrine, W. Wei, X. Li, J. Yuan, and M. Todoh, 'American Sign Language Alphabet Recognition Using Inertial Motion Capture System with Deep Learning', *Inventions*, vol. 7, no. 4, Dec. 2022, doi: 10.3390/inventions7040112.

[24] Y. Gu, H. Oku, and M. Todoh, 'American Sign Language Recognition and Translation Using Perception Neuron Wearable Inertial Motion Capture System', *Sensors*, vol. 24, no. 2, Jan. 2024, doi: 10.3390/s24020453.

[25] M. Mohammadi, P. Chandarana, J. Seekings, S. Hendrix, and R. Zand, 'Static hand gesture recognition for American sign language using neuromorphic hardware', *Neuromorphic Computing and Engineering*, vol. 2, no. 4, Dec. 2022, doi: 10.1088/2634-4386/ac94f3.

[26] S. Z. Gurbuz, A. C. Gurbuz, E. A. Malaia, D. J. Griffin, C. S. Crawford, M. M.

Rahman, E. Kurtoglu, R. Aksu, T. Macks, R. Mdrafi, ‘American Sign Language Recognition Using RF Sensing’, *IEEE Sens. J.*, vol. 21, no. 3, pp. 3763–3775, Feb. 2021, doi: 10.1109/JSEN.2020.3022376.

[27] S. Sharma and K. Kumar, ‘ASL-3DCNN: American sign language recognition technique using 3-D convolutional neural networks’, *Multimed. Tools Appl.*, vol. 80, no. 17, pp. 26319–26331, Jul. 2021, doi: 10.1007/s11042-021-10768-5.

[28] U. Hariharan, D. Devarajan, P. S. Kumar, K. Rajkumar, M. Meena, and T. Akilan, ‘Recognition of American sign language using modified deep residual CNN with modified canny edge segmentation’, *Multimed. Tools Appl.*, vol. 84, no. 31, pp. 38093–38120, Sep. 2025, doi: 10.1007/s11042-025-20663-y.

[29] I. A. Adeyanju, O. O. Bello, and M. A. Azeez, ‘Development of an American Sign Language Recognition System using Canny Edge and Histogram of Oriented Gradient’, *Nigerian Journal of Technological Development*, vol. 19, no. 3, pp. 195–205, 2022, doi: 10.4314/njtd.v19i3.2.

[30] M. A. Rahim, J. Shin, and K. S. Yun, ‘Soft Voting-based Ensemble Model for Bengali Sign Gesture Recognition’, *Annals of Emerging Technologies in Computing*, vol. 6, no. 2, pp. 41–49, 2022, doi: 10.33166/AETiC.2022.02.003.

[31] J. Shin, A. S. M. Miah, Y. Akiba, K. Hirooka, N. Hassan, and Y. S. Hwang, ‘Korean Sign Language Alphabet Recognition Through the Integration of Handcrafted and Deep Learning-Based Two-Stream Feature Extraction Approach’, *IEEE Access*, vol. 12, pp. 68303–68318, 2024, doi: 10.1109/ACCESS.2024.3399839.

[32] J. Shin, A. S. M. Miah, K. Suzuki, K. Hirooka, and M. A. M. Hasan, ‘Dynamic Korean Sign Language Recognition Using Pose Estimation Based and Attention-Based Neural Network’, *IEEE Access*, vol. 11, pp. 143501–143513, 2023, doi: 10.1109/ACCESS.2023.3343404.

[33] M. A. Bencherif, M. Algabri, M. A. Mekhtiche, M. Faisal, M. Alsulaiman, H. Mathkour, M. Al-Hammadi, and H. Ghaleb, ‘Arabic Sign Language Recognition System Using 2D Hands and Body Skeleton Data’, *IEEE Access*, vol. 9, pp. 59612–59627, 2021, doi: 10.1109/ACCESS.2021.3069714.

[34] W. Abdul, M. Alsulaiman, S. U. Amin, M. Faisal, G. Muhammad, F. R. Albogamy, M. A. Bencherif, and H. Ghaleb, ‘Intelligent real-time Arabic sign language classification using attention-based inception and BiLSTM’, *Computers and Electrical Engineering*, vol. 95, Oct. 2021, doi:

10.1016/j.compeleceng.2021.107395.

[35] G. Tharwat, A. M. Ahmed, and B. Bouallegue, ‘Arabic Sign Language Recognition System for Alphabets Using Machine Learning Techniques’, *Journal of Electrical and Computer Engineering*, vol. 2021, 2021, doi: 10.1155/2021/2995851.

[36] M. H. Ismail, S. A. Dawwd, and F. H. Ali, ‘Static hand gesture recognition of Arabic sign language by using deep CNNs’, *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 24, no. 1, pp. 178–188, Oct. 2021, doi: 10.11591/ijeecs.v24.i1.pp178-188.

[37] S. Al Ahmadi, F. Muhammad, and H. Al Dawsari, ‘Enhancing Arabic Sign Language Interpretation: Leveraging Convolutional Neural Networks and Transfer Learning’, *Mathematics*, vol. 12, no. 6, Mar. 2024, doi: 10.3390/math12060823.

[38] B. B. Al-Onazi, M. K. Nour, H. Alshahran, M. A. Elfaki, M. M. Alnfiai, R. Marzouk, M. Othman, M. M. Sharif, A. Motwakel, ‘Arabic Sign Language Gesture Classification Using Deer Hunting Optimization with Machine Learning Model’, *Computers, Materials and Continua*, vol. 75, no. 2, pp. 3413–3429, 2023, doi: 10.32604/cmc.2023.035303.

[39] A. O. Mahmoud, I. Ziedan, and A. A. Zamel, ‘Optimized Hybrid Convolution Neural Network with Machine Learning for Arabic Sign Language Recognition’, *Traitement du Signal*, vol. 41, no. 4, pp. 1835–1846, Aug. 2024, doi: 10.18280/ts.410415.

[40] T. H. Noor, A. Noor, A. F. Alharbi, A. Faisal, R. Alrashidi, A. S. Alsaedi, G. Alharbi, T. Alsanoosy, A. Alsaeedi, ‘Real-Time Arabic Sign Language Recognition Using a Hybrid Deep Learning Model’, *Sensors*, vol. 24, no. 11, Jun. 2024, doi: 10.3390/s24113683.

[41] M. A. A. Mosleh, A. Assiri, A. H. Gumaei, B. F. Alkhamees, and M. Al-Qahtani, ‘A Bidirectional Arabic Sign Language Framework Using Deep Learning and Fuzzy Matching Score’, *Mathematics*, vol. 12, no. 8, Apr. 2024, doi: 10.3390/math12081155.

[42] S. Das, M. S. Imtiaz, N. H. Neom, N. Siddique, and H. Wang, ‘A hybrid approach for Bangla sign language recognition using deep transfer learning model with random forest classifier’, *Expert Syst. Appl.*, vol. 213, Mar. 2023, doi: 10.1016/j.eswa.2022.118914.

[43] K. K. Podder, M. E. H. Chowdhury, A. M. Tahir, Z. Bin Mahbub, A. Khandakar, M. S. Hossain, and M. A. Kadir, 'Bangla Sign Language (BdSL) Alphabets and Numerals Classification Using a Deep Learning Model', *Sensors*, vol. 22, no. 2, Jan. 2022, doi: 10.3390/s22020574.

[44] R. A. Nihal, S. Rahman, N. M. Broti, and S. Ahmed Deowan, 'Bangla Sign alphabet recognition with zero-shot and transfer learning', *Pattern Recognit. Lett.*, vol. 150, pp. 84–93, Oct. 2021, doi: 10.1016/j.patrec.2021.06.020.

[45] A. B. Rebrastya, S. Adi, H. Al Fatta, W. Mega, P. Dhuhita, I. N. Fajri, M. Hanafi, and A. Yogyakarta, 'Optimization of Convolutional Neural Network Algorithm for Indonesian Sign Language Classification', *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 16, no. 9, pp. 724–733, 2025, doi: 10.14569/ijacsa.2025.0160969.

[46] M. A. Saputra and E. Rakun, 'Recognizing Indonesian sign language (Bisindo) gesture in complex backgrounds', *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 36, no. 3, p. 1583, Dec. 2024, doi: 10.11591/ijeecs.v36.i3.pp1583-1593.

[47] B. Natarajan, E. Rajalakshmi, R. Elakkiya, K. Kotecha, A. Abraham, L. A. Gabralla, and V. Subramaniyaswamy, 'Development of an End-to-End Deep Learning Framework for Sign Language Recognition, Translation, and Video Generation', *IEEE Access*, vol. 10, pp. 104358–104374, 2022, doi: 10.1109/ACCESS.2022.3210543.

[48] D. Yang, J. Li, G. Hao, Q. Chen, X. Wei, Z. Dai, Z. Hou, L. Zhang, X. Li, 'Hardware accelerator for high accuracy sign language recognition with residual network based on FPGAs', *IEICE Electronics Express*, vol. 21, no. 4, Feb. 2024, doi: 10.1587/elex.21.20230579.

[49] M. Hrúz, I. Gruber, J. Kanis, M. Boháček, M. Hlaváč, and Z. Krňoul, 'One Model is Not Enough Ensembles for Isolated Sign Language Recognition', *Sensors*, vol. 22, no. 13, Jul. 2022, doi: 10.3390/s22135043.

[50] D. Tsourounis, D. Kastaniotis, C. Theoharatos, A. Kazantzidis, and G. Economou, 'SIFT-CNN: When Convolutional Neural Networks Meet Dense SIFT Descriptors for Image and Sequence Classification', *J. Imaging*, vol. 8, no. 10, Oct. 2022, doi: 10.3390/jimaging8100256.

[51] M. Zhao, C. H. Chang, W. Xie, Z. Xie, and J. Hu, 'Cloud Shape Classification System Based on Multi-Channel CNN and Improved FDM', *IEEE Access*, vol.

8, pp. 44111–44124, 2020, doi: 10.1109/ACCESS.2020.2978090.

[52] H. Seo, A. D. Raut, C. Chen, and C. Zhang, ‘Multi-Label Classification and Automatic Damage Detection of Masonry Heritage Building through CNN Analysis of Infrared Thermal Imaging’, *Remote Sens. (Basel).*, vol. 15, no. 10, May 2023, doi: 10.3390/rs15102517.

[53] M. A. Ozdemir, D. H. Kisa, O. Guren, and A. Akan, ‘Hand gesture classification using time–frequency images and transfer learning based on CNN’, *Biomed. Signal Process. Control*, vol. 77, Aug. 2022, doi: 10.1016/j.bspc.2022.103787.

[54] M. Safran, W. Alrajhi, and S. Alfarhood, ‘DPXception: a lightweight CNN for image-based date palm species classification’, *Front. Plant Sci.*, vol. 14, 2023, doi: 10.3389/fpls.2023.1281724.

[55] H. S. Kim, J. Jung, R. Hwang, S. C. Park, S. J. Lee, G. T. Kim, and B. W. Lee, ‘Classification of PRPD Pattern in Cast-Resin Transformers Using CNN and Implementation of Explainable AI (XAI) With Grad-CAM’, *IEEE Access*, vol. 12, pp. 53623–53632, 2024, doi: 10.1109/ACCESS.2024.3365135.

[56] F. Röhrbein, A. Shoeibi, K. N. Toosi, M. Raheel Bhutta, K.-S. Hong, and H. K-s, ‘EEG-fNIRS-based hybrid image construction and classification using CNN-LSTM’, *Front. Neurorobot.*, vol. 16, Aug. 2022, doi: 10.3389/fnbot.2022.873239.

[57] C. I. Moon and O. Lee, ‘Skin Microstructure Segmentation and Aging Classification Using CNN-Based Models’, *IEEE Access*, vol. 10, pp. 4948–4956, 2022, doi: 10.1109/ACCESS.2021.3140031.

[58] J. Espejel-Cabrera, J. Cervantes, F. García-Lamont, J. S. Ruiz Castilla, and L. D. Jalili, ‘Mexican sign language segmentation using color based neuronal networks to detect the individual skin color’, *Expert Syst. Appl.*, vol. 183, Nov. 2021, doi: 10.1016/j.eswa.2021.115295.

[59] A. Mohanty, K. Roy, and R. R. Sahay, ‘Robust static hand gesture recognition: harnessing sparsity of deeply learned features’, *Visual Computer*, vol. 40, no. 9, pp. 6507–6531, Sep. 2024, doi: 10.1007/s00371-023-03179-0.

[60] A. M. Buttar, U. Ahmad, A. H. Gumaei, A. Assiri, M. A. Akbar, and B. F. Alkhamees, ‘Deep Learning in Sign Language Recognition: A Hybrid Approach for the Recognition of Static and Dynamic Signs’, *Mathematics*, vol. 11, no. 17, Sep. 2023, doi: 10.3390/math11173729.

[61] M. K. Benkaddour, ‘CNN based features extraction for age estimation and gender classification’, *Informatica (Slovenia)*, vol. 45, no. 5, pp. 697–703, 2021, doi: 10.31449/INF.V45I5.3262.

[62] A. Tyagi and S. Bansal, ‘Hybrid FiST_CNN approach for Feature Extraction for Vision-Based Indian Sign Language Recognition’, *The International Arab Journal of Information Technology*, vol. 19, no. 3, 2022, doi: 10.34028/iajit/19/3/15.

[63] M. Kakizaki, A. S. M. Miah, K. Hirooka, and J. Shin, ‘Dynamic Japanese Sign Language Recognition Throw Hand Pose Estimation Using Effective Feature Extraction and Classification Approach’, *Sensors*, vol. 24, no. 3, Feb. 2024, doi: 10.3390/s24030826.

[64] P. Pannattee, W. Kumwilaisak, C. Hansakunbuntheung, N. Thatphithakkul, and C. C. J. Kuo, ‘American Sign language fingerspelling recognition in the wild with spatio temporal feature extraction and multi-task learning’, *Expert Syst. Appl.*, vol. 243, Jun. 2024, doi: 10.1016/j.eswa.2023.122901.

[65] S. R. Waheed, S. M. Saadi, M. S. M. Rahim, N. M. Suaib, F. H. Najjar, M. M. Adnan, and A. A. Salim, ‘Melanoma Skin Cancer Classification based on CNN Deep Learning Algorithms’, *Malaysian Journal of Fundamental and Applied Sciences*, vol. 19, no. 3, pp. 299–305, Jan. 2023, doi: 10.11113/mjfas.v19n3.2900.

[66] T. K. Dutta, D. R. Nayak, and Y. D. Zhang, ‘ARM-Net: Attention-guided residual multiscale CNN for multiclass brain tumor classification using MR images’, *Biomed. Signal Process. Control*, vol. 87, Jan. 2024, doi: 10.1016/j.bspc.2023.105421.

[67] K. Dimililer and B. Sekeroglu, ‘Skin Lesion Classification Using CNN-based Transfer Learning Model’, *Gazi University Journal of Science*, vol. 36, no. 2, pp. 660–673, Jun. 2023, doi: 10.35378/gujs.1063289.

[68] P. Moisés De Sousa, P. Cunha Carneiro, M. O. Mariane, M. P. Gabrielle, A. da C. J. Carlos, V. de M. Luis, M. Christian, M. M da S. Ana, C. P. Ana, ‘COVID-19 classification in X-ray chest images using a new convolutional neural network: CNN-COVID’, *Research on Biomedical Engineering*, 2022, doi: 10.1007/s42600-020-00120-5.

[69] K. G. Satheeshkumar, V. Arunachalam, and S. Deepika, ‘Hand-held GPU accelerated device for multiclass classification of X-ray images using CNN

model', *Microprocess. Microsyst.*, vol. 106, Apr. 2024, doi: 10.1016/j.micpro.2024.105046.

[70] D. S. Rao, C. Ramesh Babu, V. S. Kiran, N. Rajasekhar, K. Srinivas, P. S. Akshay, G. S. Mohan, and B. L. Bharadwaj, 'Plant disease classification using deep bilinear cnn', *Intelligent Automation and Soft Computing*, vol. 31, no. 1, pp. 161–176, 2022, doi: 10.32604/IASC.2022.017706.

[71] K. Hameed, D. Chai, and A. Rassau, 'A sample weight and adaboost CNN-based coarse to fine classification of fruit and vegetables at a supermarket self-checkout', *Applied Sciences (Switzerland)*, vol. 10, no. 23, pp. 1–18, Dec. 2020, doi: 10.3390/app10238667.

[72] R. Patil, Y. Sinkar, A. Ruke, H. Kulkarni, and O. Kadam, 'Smart Agri-Advisor: Integrating Chatbot Technology with CNN-Based Crop Disease Classification for Enhanced Agricultural Decision-Making', *International Journal of Engineering Trends and Technology*, vol. 72, no. 7, pp. 375–380, Jul. 2024, doi: 10.14445/22315381/IJETT-V72I7P141.

[73] G. S. R. Satyanarayana, P. Deshmukh, and S. K. Das, 'Vehicle detection and classification with spatio-temporal information obtained from CNN', *Displays*, vol. 75, Dec. 2022, doi: 10.1016/j.displa.2022.102294.

[74] H. Gao, B. Cheng, J. Wang, K. Li, J. Zhao, and D. Li, 'Object Classification Using CNN-Based Fusion of Vision and LIDAR in Autonomous Vehicle Environment', *IEEE Trans. Industr. Inform.*, vol. 14, no. 9, pp. 4224–4230, Sep. 2018, doi: 10.1109/TII.2018.2822828.

[75] A. O. Alaoui, O. El Bahi, M. R. Fethi, O. Farhaoui, A. El Allaoui, and Y. Farhaoui, 'Pre-trained CNNs: Evaluating Emergency Vehicle Image Classification', *Data and Metadata*, vol. 2, Jan. 2023, doi: 10.56294/dm2023153.

[76] Y. Pei, Y. Huang, Q. Zou, X. Zhang, and S. Wang, 'Effects of Image Degradation and Degradation Removal to CNN-Based Image Classification', *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 4, pp. 1239–1253, Apr. 2021, doi: 10.1109/TPAMI.2019.2950923.

[77] M. Zhang, W. Li, R. Tao, H. Li, and Q. Du, 'Information Fusion for Classification of Hyperspectral and LiDAR Data Using IP-CNN', *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, 2022, doi: 10.1109/TGRS.2021.3093334.

[78] G. Wang, H. Ruser, J. Schade, J. Passig, T. Adam, G. Dollinger, and R. Zimmermann, '1D-CNN Network Based Real-Time Aerosol Particle Classification with Single-Particle Mass Spectrometry', *IEEE Sens. Lett.*, vol. 7, no. 11, Nov. 2023, doi: 10.1109/LSENS.2023.3315554.

[79] Z. Mei, K. Ivanov, G. Zhao, Y. Wu, M. Liu, and L. Wang, 'Foot type classification using sensor-enabled footwear and 1D-CNN', *Measurement (Lond).*, vol. 165, Dec. 2020, doi: 10.1016/j.measurement.2020.108184.

[80] M. Djouadi and M. K. Kholladi, 'Improving Street View Image Classification Using Pre-trained CNN Model Extracted Features', *Periodica polytechnica Electrical engineering and computer science*, vol. 66, no. 4, pp. 370–379, 2022, doi: 10.3311/PPee.19961.

[81] Y. Lu, K. Xie, G. Xu, H. Dong, C. Li, and T. Li, 'MTFC: A Multi-GPU Training Framework for Cube-CNN-Based Hyperspectral Image Classification', *IEEE Trans. Emerg. Top. Comput.*, vol. 9, no. 4, pp. 1738–1752, 2021, doi: 10.1109/TETC.2020.3016978.

[82] X. Chen, L. Xie, J. Wu, and Q. Tian, 'Cyclic CNN: Image Classification with Multiscale and Multilocation Contexts', *IEEE Internet Things J.*, vol. 8, no. 9, pp. 7466–7475, May 2021, doi: 10.1109/JIOT.2020.3038644.

[83] G. A. Mesías-Ruiz, I. Borra-Serrano, J. M. Peña, A. I. de Castro, C. Fernández-Quintanilla, and J. Dorado, 'Weed species classification with UAV imagery and standard CNN models: Assessing the frontiers of training and inference phases', *Crop Protection*, vol. 182, Aug. 2024, doi: 10.1016/j.cropro.2024.106721.

[84] M. F. Ahamed, M. Nahiduzzaman, M. R. Islam, M. Naznine, M. Arselene Ayari, A. Khandakar, and J. Haider, 'Detection of various gastrointestinal tract diseases through a deep learning method with ensemble ELM and explainable AI', *Expert Syst. Appl.*, vol. 256, Dec. 2024, doi: 10.1016/j.eswa.2024.124908.

[85] M. S. Islam, S. Sultana, F. Al Farid, M. N. Islam, M. Rashid, B. S. Bari, N. Hashim, and M. N. Husen, 'Multimodal Hybrid Deep Learning Approach to Detect Tomato Leaf Disease Using Attention Based Dilated Convolution Feature Extractor with Logistic Regression Classification', *Sensors*, vol. 22, no. 16, Aug. 2022, doi: 10.3390/s22166079.

[86] R. Alizadehsani, D. Sharifrazi, N. H. Izadi, J. H. Joloudari, A. Shoelbi, J. M. Gorriz, S. Hussain, J. E. Arco, Z. A. Sani, F. Khozeimeh, A. Khosravi, S.

Nahavandi, S. M. S. Islam, U. J. Acharya, 'Uncertainty-aware semi-supervised method using large unlabeled and limited labeled COVID-19 data', *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 17, no. 3s, Oct. 2021, doi: 10.1145/3462635.

[87] D. N. Le, V. S. Parvathy, D. Gupta, A. Khanna, J. J. P. C. Rodrigues, and K. Shankar, 'IoT enabled depthwise separable convolution neural network with deep support vector machine for COVID-19 diagnosis and classification', *International Journal of Machine Learning and Cybernetics*, vol. 12, no. 11, pp. 3235–3248, Nov. 2021, doi: 10.1007/s13042-020-01248-7.

[88] P. Zhou, Y. Cao, M. Li, Y. Ma, C. Chen, X. Gan, J. Wu, X. Lv, C. Chen, 'HCCANet: histopathological image grading of colorectal cancer using CNN based on multichannel fusion attention mechanism', *Sci. Rep.*, vol. 12, no. 1, Dec. 2022, doi: 10.1038/s41598-022-18879-1.

[89] A. Das, 'Adaptive UNet-based Lung Segmentation and Ensemble Learning with CNN-based Deep Features for Automated COVID-19 Diagnosis', *Multimed. Tools Appl.*, vol. 81, no. 4, pp. 5407–5441, Feb. 2022, doi: 10.1007/s11042-021-11787-y.

[90] P. K. Balasubramanian, W. C. Lai, G. H. Seng, C. Kavitha, and J. Selvaraj, 'APESTNet with Mask R-CNN for Liver Tumor Segmentation and Classification', *Cancers (Basel).*, vol. 15, no. 2, Jan. 2023, doi: 10.3390/cancers15020330.

[91] M. Y. B. Murthy, A. Koteswararao, and M. S. Babu, 'Adaptive fuzzy deformable fusion and optimized CNN with ensemble classification for automated brain tumor diagnosis', *Biomed. Eng. Lett.*, vol. 12, no. 1, pp. 37–58, Feb. 2022, doi: 10.1007/s13534-021-00209-5.

[92] A. Maqsood, M. S. Farid, M. H. Khan, and M. Grzegorzek, 'Deep malaria parasite detection in thin blood smear microscopic images', *Applied Sciences (Switzerland)*, vol. 11, no. 5, pp. 1–19, Mar. 2021, doi: 10.3390/app11052284.

[93] T. Vaiyapuri, P. Balaji, S. Shridevi, H. Alaskar, and Z. Sbai, 'Computational Intelligence-Based Melanoma Detection and Classification Using Dermoscopic Images', *Comput. Intell. Neurosci.*, vol. 2022, 2022, doi: 10.1155/2022/2370190.

[94] B. Praveen and V. Menon, 'Study of Spatial-Spectral Feature Extraction Frameworks with 3-D Convolutional Neural Network for Robust Hyperspectral

Imagery Classification', *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 14, pp. 1717–1727, 2021, doi: 10.1109/JSTARS.2020.3046414.

[95] D. S. Breland, S. B. Skriubakken, A. Dayal, A. Jha, P. K. Yalavarthy, and L. R. Cenkeramaddi, 'Deep Learning-Based Sign Language Digits Recognition from Thermal Images with Edge Computing System', *IEEE Sens. J.*, vol. 21, no. 9, pp. 10445–10453, May 2021, doi: 10.1109/JSEN.2021.3061608.

[96] R. Pitonak, J. Mucha, L. Dobis, M. Javorka, and M. Marusin, 'CloudSatNet-1: FPGA-Based Hardware-Accelerated Quantized CNN for Satellite On-Board Cloud Coverage Classification', *Remote Sens. (Basel).*, vol. 14, no. 13, Jul. 2022, doi: 10.3390/rs14133180.

[97] N. S. Y. Tan, M. L. Tham, S. Y. Chua, and Y. L. Lee, 'Accurate and Fast Classification of Natural Disasters using CNN-LSTM and Inference Acceleration', *Journal of Communications Software and Systems*, vol. 20, no. 1, pp. 58–68, 2024, doi: 10.24138/jcomss-2023-0129.

[98] S. In Cho, 'Vision-based people counter using CNN-Based event classification', *IEEE Trans. Instrum. Meas.*, vol. 69, no. 8, pp. 5308–5315, Aug. 2020, doi: 10.1109/TIM.2019.2959853.

[99] N. Sabor, G. Gendy, H. Mohammed, G. Wang, and Y. Lian, 'Robust Arrhythmia Classification Based on QRS Detection and a Compact 1D-CNN for Wearable ECG Devices', *IEEE J. Biomed. Health Inform.*, vol. 26, no. 12, pp. 5918–5929, Dec. 2022, doi: 10.1109/JBHI.2022.3207456.

[100] Q. Li, J. Lei, C. Ren, Z. Peng, and J. Hong, 'Research on sports image classification method based on SE-RES-CNN model', *Sci. Rep.*, vol. 14, no. 1, Dec. 2024, doi: 10.1038/s41598-024-69965-5.

[101] E. Arı and E. Taçgın, 'NF-EEG: A generalized CNN model for multi class EEG motor imagery classification without signal preprocessing for brain computer interfaces', *Biomed. Signal Process. Control*, vol. 92, Jun. 2024, doi: 10.1016/j.bspc.2024.106081.

[102] R. Avanzato and F. Beritelli, 'A CNN-based differential image processing approach for rainfall classification', *Advances in Science, Technology and Engineering Systems*, vol. 5, no. 4, pp. 438–444, 2020, doi: 10.25046/aj050452.

[103] M. Jaiswal, V. Sharma, A. Sharma, S. Saini, and R. Tomar, 'Quantized CNN-based efficient hardware architecture for real-time hand gesture recognition', *Microelectronics J.*, vol. 151, Sep. 2024, doi: 10.1016/j.mejo.2024.106345.

[104] L. Peng, J. Yang, Z. Chen, L. Yan, X. Jiao, J. Xiao, L. Zhou, L. Chang, Y. Long, J. Zhou, 'A High Accuracy and Low Power CNN-Based Environmental Sound Classification Processor', *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 12, pp. 4865–4876, Dec. 2023, doi: 10.1109/TCSI.2023.3299823.

[105] C. Zhang, J. Li, P. Guo, Q. Li, X. Zhang, and X. Wang, 'A configurable hardware-efficient ECG classification inference engine based on CNN for mobile healthcare applications', *Microelectronics Journal* , vol. 141, Nov. 2023, doi: 10.1016/j.mejo.2023.105969.

[106] A. Ghani, R. Hodeify, C. H. See, S. Keates, D. J. Lee, and A. Bouridane, 'Computer Vision-Based Kidney's (HK-2) Damaged Cells Classification with Reconfigurable Hardware Accelerator (FPGA)', *Electronics (Switzerland)*, vol. 11, no. 24, Dec. 2022, doi: 10.3390/electronics11244234.

[107] V. P. Saxena, P. P. Shah, M. Kumhar, J. Bhatia, P. Saxena, and S. Tanwar, 'Hybrid Convolutional Neural Mixed Approached Model for Incorporating Sign Language Features', *SN Comput. Sci.*, vol. 6, no. 6, Aug. 2025, doi: 10.1007/s42979-025-04131-w.

[108] Z. Wang, T. Zhao, J. Ma, H. Chen, K. Liu, H. Shao, Q. Wang, and J. Ren, 'Hear Sign Language: A Real-Time End-to-End Sign Language Recognition System', *IEEE Trans. Mob. Comput.*, vol. 21, no. 7, pp. 2398–2410, Jul. 2022, doi: 10.1109/TMC.2020.3038303.

[109] M. Vilasini and P. Ramamoorthy, 'CNN approaches for classification of Indian leaf species using smartphones', *Computers, Materials and Continua*, vol. 62, no. 3, pp. 1445–1472, 2020, doi: 10.32604/cmc.2020.08857.

[110] A. Osman Mahmoud, I. Ziedan, and A. Ahmed Zamel, 'An Efficient Convolutional Neural Network Classification Model for Several Sign Language Alphabets', *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 14, no. 9, pp. 1040–1050, 2023, doi: 10.14569/IJACSA.2023.01409108.

[111] M. M. Damaneh, F. Mohanna, and P. Jafari, 'Static hand gesture recognition in sign language based on convolutional neural network with feature extraction method using ORB descriptor and Gabor filter', *Expert Syst. Appl.*, vol. 211, Jan. 2023, doi: 10.1016/j.eswa.2022.118559.

[112] R. Tomari, N. Syahirah Razali, N. Farhana Santosa, A. Abdul Kadir, M. Fahrul

Hassan, B. Pahat, and J. Malaysia, ‘Reverse Vending Machine Item Verification Module using Classification and Detection Model of CNN’, *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 12, no. 10, pp. 401–407, 2021, [Online]. Available: www.ijacsa.thesai.org

# APPENDICES

# APPENDIX 1

Gaussian filter module in Quartus

```
module gaussian_filter (
    input clk,
    input rst_n,
    input [23:0] pixel_in,
    input pixel_valid,
    input [12:0] x_pos,
    input [12:0] y_pos,
    output reg [7:0] pixel_out,
    output reg pixel_valid_out
);

    // FIXED: Use temporary wires for bit slicing
    wire [7:0] red   = pixel_in[23:16];
    wire [7:0] green = pixel_in[15:8];
    wire [7:0] blue  = pixel_in[7:0];

    // USE EXACT SAME GRAYSCALE AS YOUR RGB2GRAY
    wire [7:0] grayscale_pixel = red[7:2] +      // R/4 = 0.25R
                                 green[7:1] +    // G/2 = 0.5G
                                 green[7:3] +    // G/8 = 0.125G
                                 blue[7:3];      // B/8 = 0.125B

    // Gaussian kernel (3x3) - Integer approximation
    parameter [2:0] KERNEL_00 = 1, KERNEL_01 = 2, KERNEL_02 = 1;
    parameter [2:0] KERNEL_10 = 2, KERNEL_11 = 4, KERNEL_12 = 2;
    parameter [2:0] KERNEL_20 = 1, KERNEL_21 = 2, KERNEL_22 = 1;
    localparam KERNEL_SUM = 16;

    // Line buffers
    reg [7:0] line_buffer_0 [0:639];
    reg [7:0] line_buffer_1 [0:639];
    reg [7:0] window [0:2][0:2];
    reg [2:0] valid_pipeline;

    // Window formation with border handling
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
        begin : init_loop
            integer i, j;
            for (i=0; i<3; i=i+1) for (j=0; j<3; j=j+1)
                window[i][j] <= 8'd0;
        end
        valid_pipeline <= 3'd0;
    end
    else if (pixel_valid) begin
        // Update line buffers
        if (y_pos > 0) begin
            line_buffer_1[x_pos] <= line_buffer_0[x_pos];
            line_buffer_0[x_pos] <= grayscale_pixel;
        end

        // Build 3x3 window with border replication
        if (x_pos >= 1 && y_pos >= 1) begin
            // Top row (y-2)
            window[0][0] <= (x_pos == 1 || y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos-1];
            window[0][1] <= (y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos];
            window[0][2] <= (x_pos == 13'd639 || y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos+1];

            // Middle row (y-1)
            window[1][0] <= (x_pos == 1) ? grayscale_pixel : line_buffer_0[x_pos-1];
            window[1][1] <= grayscale_pixel;
            window[1][2] <= (x_pos == 13'd639) ? grayscale_pixel : line_buffer_0[x_pos+1];
```

```
            // Bottom row (y)
            window[2][0] <= (x_pos == 1 || y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;
            window[2][1] <= (y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;
            window[2][2] <= (x_pos == 13'd639 || y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;
        end

        valid_pipeline <= {valid_pipeline[1:0], (x_pos >= 2 && y_pos >= 2)};
    end
end

// Convolution with single multiplier (saves DSP blocks)
reg [15:0] accumulator;
reg [3:0] state;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        accumulator <= 16'd0;
        state <= 4'd0;
        pixel_out <= 8'd0;
        pixel_valid_out <= 1'b0;
    end
    else begin
        pixel_valid_out <= valid_pipeline[2];

        if (valid_pipeline[1]) begin
            case(state)
                0: begin accumulator <= window[0][0] * KERNEL_00; state <= 4'd1; end
                1: begin accumulator <= accumulator + (window[0][1] * KERNEL_01); state <= 4'd2; end
                2: begin accumulator <= accumulator + (window[0][2] * KERNEL_02); state <= 4'd3; end
                3: begin accumulator <= accumulator + (window[1][0] * KERNEL_10); state <= 4'd4; end
                4: begin accumulator <= accumulator + (window[1][1] * KERNEL_11); state <= 4'd5; end
                5: begin accumulator <= accumulator + (window[1][2] * KERNEL_12); state <= 4'd6; end
                6: begin accumulator <= accumulator + (window[2][0] * KERNEL_20); state <= 4'd7; end
                7: begin accumulator <= accumulator + (window[2][1] * KERNEL_21); state <= 4'd8; end
                8: begin accumulator <= accumulator + (window[2][2] * KERNEL_22); state <= 4'd9; end
                9: begin
                    pixel_out <= (accumulator / KERNEL_SUM);
                    state <= 4'd0;
                end
            endcase
        end
    end
end

endmodule
```

Median filter module in Quartus

```
module median_filter (
    input clk,
    input rst_n,
    input [23:0] pixel_in,
    input pixel_valid,
    input [12:0] x_pos,
    input [12:0] y_pos,
    output reg [7:0] pixel_out,
    output reg pixel_valid_out
);

    // FIXED: Use temporary wires
    wire [7:0] red   = pixel_in[23:16];
    wire [7:0] green = pixel_in[15:8];
    wire [7:0] blue  = pixel_in[7:0];

    // USE EXACT SAME GRAYSCALE AS YOUR RGB2GRAY
    wire [7:0] grayscale_pixel = red[7:2] +     // R/4
                                 green[7:1] +    // G/2
                                 green[7:3] +    // G/8
                                 blue[7:3];      // B/8

    // Line buffers
    reg [7:0] line_buffer_0 [0:639];
    reg [7:0] line_buffer_1 [0:639];
    reg [7:0] window [0:2][0:2];
    reg [2:0] valid_pipeline;

    // Window formation (same as Gaussian)
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            begin : init_loop
                integer i, j;
                for (i=0; i<3; i=i+1) for (j=0; j<3; j=j+1)
                    window[i][j] <= 8'd0;
            end
            valid_pipeline <= 3'd0;
        end
        else if (pixel_valid) begin
            if (y_pos > 0) begin
                line_buffer_1[x_pos] <= line_buffer_0[x_pos];
                line_buffer_0[x_pos] <= grayscale_pixel;
            end

            if (x_pos >= 1 && y_pos >= 1) begin
                window[0][0] <= (x_pos == 1 || y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos-1];
                window[0][1] <= (y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos];
                window[0][2] <= (x_pos == 13'd639 || y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos+1];

                window[1][0] <= (x_pos == 1) ? grayscale_pixel : line_buffer_0[x_pos-1];
                window[1][1] <= grayscale_pixel;
                window[1][2] <= (x_pos == 13'd639) ? grayscale_pixel : line_buffer_0[x_pos+1];

                window[2][0] <= (x_pos == 1 || y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;
                window[2][1] <= (y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;
                window[2][2] <= (x_pos == 13'd639 || y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;
            end

            valid_pipeline <= {valid_pipeline[1:0], (x_pos >= 2 && y_pos >= 2)};
        end
    end

    // Optimized median calculation
    reg [7:0] values [0:8];
    reg [2:0] sort_stage;
```

```
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            begin : init_values
                integer i;
                for (i=0; i<9; i=i+1) values[i] <= 8'd0;
            end
            sort_stage <= 3'd0;
            pixel_out <= 8'd0;
            pixel_valid_out <= 1'b0;
        end
        else begin
            pixel_valid_out <= valid_pipeline[2];

            if (valid_pipeline[1]) begin
                case(sort_stage)
                    0: begin
                        // Load window into 1D array
                        values[0] <= window[0][0]; values[1] <= window[0][1]; values[2] <= window[0][2];
                        values[3] <= window[1][0]; values[4] <= window[1][1]; values[5] <= window[1][2];
                        values[6] <= window[2][0]; values[7] <= window[2][1]; values[8] <= window[2][2];
                        sort_stage <= 3'd1;
                    end
                    1: begin
                        // First sort pass
                        if (values[0] > values[1]) begin values[0] <= values[1]; values[1] <= values[0]; end
                        if (values[2] > values[3]) begin values[2] <= values[3]; values[3] <= values[2]; end
                        if (values[4] > values[5]) begin values[4] <= values[5]; values[5] <= values[4]; end
                        if (values[6] > values[7]) begin values[6] <= values[7]; values[7] <= values[6]; end
                        sort_stage <= 3'd2;
                    end
                    2: begin
                        // Second pass
                        if (values[1] > values[2]) begin values[1] <= values[2]; values[2] <= values[1]; end
                        if (values[3] > values[4]) begin values[3] <= values[4]; values[4] <= values[3]; end
                        if (values[5] > values[6]) begin values[5] <= values[6]; values[6] <= values[5]; end
                        if (values[7] > values[8]) begin values[7] <= values[8]; values[8] <= values[7]; end
                        sort_stage <= 3'd3;
                    end
                    3: begin
                        // Third pass - enough for median
                        if (values[0] > values[1]) begin values[0] <= values[1]; values[1] <= values[0]; end
                        if (values[2] > values[3]) begin values[2] <= values[3]; values[3] <= values[2]; end
                        if (values[4] > values[5]) begin values[4] <= values[5]; values[5] <= values[4]; end
                        if (values[6] > values[7]) begin values[6] <= values[7]; values[7] <= values[6]; end
                        sort_stage <= 3'd4;
                    end
                    4: begin
                        // Output median (5th element)
                        pixel_out <= values[4];
                        sort_stage <= 3'd0;
                    end
                endcase
            end
        end
    end

endmodule
```

Bilateral filter module in Quartus

```
module bilateral_filter (
    input clk,
    input rst_n,
    input [23:0] pixel_in,
    input pixel_valid,
    input [12:0] x_pos,
    input [12:0] y_pos,
    output reg [7:0] pixel_out,
    output reg pixel_valid_out
);

    // FIXED: Use temporary wires
    wire [7:0] red   = pixel_in[23:16];
    wire [7:0] green = pixel_in[15:8];
    wire [7:0] blue  = pixel_in[7:0];

    // USE EXACT SAME GRAYSCALE AS YOUR RGB2GRAY
    wire [7:0] grayscale_pixel = red[7:2] +     // R/4
                                 green[7:1] +    // G/2
                                 green[7:3] +    // G/8
                                 blue[7:3];      // B/8


    // Spatial weights (Gaussian approximation)
    parameter [7:0] SPATIAL_00 = 8, SPATIAL_01 = 6, SPATIAL_02 = 4;
    parameter [7:0] SPATIAL_10 = 6, SPATIAL_11 = 9, SPATIAL_12 = 6;
    parameter [7:0] SPATIAL_20 = 4, SPATIAL_21 = 6, SPATIAL_22 = 4;

    // Line buffers and window
    reg [7:0] line_buffer_0 [0:639];
    reg [7:0] line_buffer_1 [0:639];
    reg [7:0] window [0:2][0:2];
    reg [7:0] center_pixel;
    reg [2:0] valid_pipeline;

    // Window formation
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            begin : init_loop
                integer i, j;
                for (i=0; i<3; i=i+1) for (j=0; j<3; j=j+1)
                    window[i][j] <= 8'd0;
            end
            center_pixel <= 8'd0;
            valid_pipeline <= 3'd0;
        end
        else if (pixel_valid) begin
            if (y_pos > 0) begin
                line_buffer_1[x_pos] <= line_buffer_0[x_pos];
                line_buffer_0[x_pos] <= grayscale_pixel;
            end

            if (x_pos >= 1 && y_pos >= 1) begin
                window[0][0] <= (x_pos == 1 || y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos-1];
                window[0][1] <= (y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos];
                window[0][2] <= (x_pos == 13'd639 || y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos+1];

                window[1][0] <= (x_pos == 1) ? grayscale_pixel : line_buffer_0[x_pos-1];
                window[1][1] <= grayscale_pixel;
                window[1][2] <= (x_pos == 13'd639) ? grayscale_pixel : line_buffer_0[x_pos+1];

                window[2][0] <= (x_pos == 1 || y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;
                window[2][1] <= (y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;
                window[2][2] <= (x_pos == 13'd639 || y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;

                center_pixel <= grayscale_pixel;
            end

            valid_pipeline <= {valid_pipeline[1:0], (x_pos >= 2 && y_pos >= 2)};
        end
    end

    // Bilateral calculation
    reg [15:0] weighted_sum, weight_sum;
    reg [3:0] calc_state;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            weighted_sum <= 16'd0;
            weight_sum <= 16'd0;
            calc_state <= 4'd0;
            pixel_out <= 8'd0;
            pixel_valid_out <= 1'b0;
        end
        else begin
            pixel_valid_out <= valid_pipeline[2];

            if (valid_pipeline[1]) begin
                case(calc_state)
                    0: begin weighted_sum <= 16'd0; weight_sum <= 16'd0; calc_state <= 4'd1; end
                    1: begin process_pixel(0,0,SPATIAL_00); calc_state <= 4'd2; end
                    2: begin process_pixel(0,1,SPATIAL_01); calc_state <= 4'd3; end
                    3: begin process_pixel(0,2,SPATIAL_02); calc_state <= 4'd4; end
                    4: begin process_pixel(1,0,SPATIAL_10); calc_state <= 4'd5; end
                    5: begin process_pixel(1,1,SPATIAL_11); calc_state <= 4'd6; end
                    6: begin process_pixel(1,2,SPATIAL_12); calc_state <= 4'd7; end
                    7: begin process_pixel(2,0,SPATIAL_20); calc_state <= 4'd8; end
                    8: begin process_pixel(2,1,SPATIAL_21); calc_state <= 4'd9; end
                    9: begin process_pixel(2,2,SPATIAL_22); calc_state <= 4'd10; end
                    10: begin
                        if (weight_sum != 0)
                            pixel_out <= (weighted_sum / weight_sum);
                        else
                            pixel_out <= center_pixel;
                        calc_state <= 4'd0;
                    end
                endcase
            end
        end
    end

    // Helper task for pixel processing
    task process_pixel;
        input [1:0] i, j;
        input [7:0] spatial_w;
        reg [7:0] range_w;
        reg [15:0] bilateral_w;
        begin
            range_w = range_weight(center_pixel, window[i][j]);
            bilateral_w = spatial_w * range_w;
            weighted_sum <= weighted_sum + (window[i][j] * bilateral_w);
            weight_sum <= weight_sum + bilateral_w;
        end
    endtask

    // Range weight LUT (Gaussian approximation)
    function [7:0] range_weight;
        input [7:0] center, neighbor;
        reg [7:0] diff;
        begin
            diff = (center > neighbor) ? (center - neighbor) : (neighbor - center);
            case(diff)
                8'd0: range_weight = 8'd255;
                8'd1: range_weight = 8'd253;
                8'd2: range_weight = 8'd248;
                8'd3: range_weight = 8'd240;
                8'd4: range_weight = 8'd230;
                8'd5: range_weight = 8'd218;
                8'd10: range_weight = 8'd165;
                8'd15: range_weight = 8'd118;
                8'd20: range_weight = 8'd82;
                8'd25: range_weight = 8'd55;
                8'd30: range_weight = 8'd36;
                8'd40: range_weight = 8'd15;
                8'd50: range_weight = 8'd6;
                default: range_weight = 8'd0;
            endcase
        end
    endfunction

endmodule
```

Sobel filter module in Quartus

```
module sobel_filter (
    input clk,
    input rst_n,
    input [23:0] pixel_in,
    input pixel_valid,
    input [12:0] x_pos,
    input [12:0] y_pos,
    output reg [7:0] pixel_out,
    output reg pixel_valid_out
);

    // FIXED: Use temporary wires
    wire [7:0] red   = pixel_in[23:16];
    wire [7:0] green = pixel_in[15:8];
    wire [7:0] blue  = pixel_in[7:0];

    // USE EXACT SAME GRAYSCALE AS YOUR RGB2GRAY
    wire [7:0] grayscale_pixel = red[7:2] +      // R/4
                                 green[7:1] +    // G/2
                                 green[7:3] +    // G/8
                                 blue[7:3];      // B/8

    // Line buffers and window
    reg [7:0] line_buffer_0 [0:639];
    reg [7:0] line_buffer_1 [0:639];
    reg [7:0] window [0:2][0:2];
    reg [2:0] valid_pipeline;

    // Window formation
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            begin : init_loop
                integer i, j;
                for (i=0; i<3; i=i+1) for (j=0; j<3; j=j+1)
                    window[i][j] <= 8'd0;
            end
            valid_pipeline <= 3'd0;
        end
        else if (pixel_valid) begin
            if (y_pos > 0) begin
                line_buffer_1[x_pos] <= line_buffer_0[x_pos];
                line_buffer_0[x_pos] <= grayscale_pixel;
            end

            if (x_pos >= 1 && y_pos >= 1) begin
                window[0][0] <= (x_pos == 1 || y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos-1];
                window[0][1] <= (y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos];
                window[0][2] <= (x_pos == 13'd639 || y_pos == 1) ? grayscale_pixel : line_buffer_1[x_pos+1];

                window[1][0] <= (x_pos == 1) ? grayscale_pixel : line_buffer_0[x_pos-1];
                window[1][1] <= grayscale_pixel;
                window[1][2] <= (x_pos == 13'd639) ? grayscale_pixel : line_buffer_0[x_pos+1];

                window[2][0] <= (x_pos == 1 || y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;
                window[2][1] <= (y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;
                window[2][2] <= (x_pos == 13'd639 || y_pos == 13'd479) ? grayscale_pixel : grayscale_pixel;
            end

            valid_pipeline <= {valid_pipeline[1:0], (x_pos >= 2 && y_pos >= 2)};
        end
    end

    // Gradient calculation
    reg signed [10:0] gx, gy;
    reg [3:0] calc_state;


    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            gx <= 11'd0;
            gy <= 11'd0;
            calc_state <= 4'd0;
            pixel_out <= 8'd0;
            pixel_valid_out <= 1'b0;
        end
        else begin
            pixel_valid_out <= valid_pipeline[2];

            if (valid_pipeline[1]) begin
                case(calc_state)
                    0: begin gx <= 11'd0; gy <= 11'd0; calc_state <= 4'd1; end
                    // Gx gradient: right - left
                    1: begin gx <= gx + (window[0][2] - window[0][0]); calc_state <= 4'd2; end
                    2: begin gx <= gx + ((window[1][2] - window[1][0]) << 1); calc_state <= 4'd3; end
                    3: begin gx <= gx + (window[2][2] - window[2][0]); calc_state <= 4'd4; end
                    // Gy gradient: bottom - top
                    4: begin gy <= gy + (window[2][0] - window[0][0]); calc_state <= 4'd5; end
                    5: begin gy <= gy + ((window[2][1] - window[0][1]) << 1); calc_state <= 4'd6; end
                    6: begin gy <= gy + (window[2][2] - window[0][2]); calc_state <= 4'd7; end
                    7: begin
                        // Magnitude approximation: |Gx| + |Gy|
                        pixel_out <= abs(gx) + abs(gy)) > 255 ? 8'd255 : (abs(gx) + abs(gy));
                        calc_state <= 4'd0;
                    end
                endcase
            end
        end
    end

    // Absolute value function
    function [10:0] abs(input signed [10:0] val);
        abs = val[10] ? -val : val;
    endfunction

endmodule
```

Co-design Gaussian filter script in MATLAB

gaussian_filter_matlab.m

```
function output = gaussian_filter_matlab(input)

    [h, w] = size(input);

    % Gaussian kernel
    kernel = [1, 2, 1;
              2, 4, 2;
              1, 2, 1] / 16;

    % Add border padding
    padded = padarray(double(input), [1, 1], 'replicate');
    output = zeros(h, w, 'like', input);

    for i = 1:h
        for j = 1:w
            % Extract 3x3 window
            window = double(padded(i:i+2, j:j+2));

            % Convolution
            result = sum(sum(window .* kernel));

            % Output
            output(i, j) = result;
        end
    end
end
```

Co-design Median filter script in MATLAB

median_filter_matlab.m

```
function output = median_filter_matlab(input)

    [h, w] = size(input);

    % Add border padding
    padded = padarray(input, [1, 1], 'replicate');
    output = zeros(h, w, 'like', input);

    for i = 1:h
        for j = 1:w
            % Extract 3x3 window
            window = padded(i:i+2, j:j+2);

            % Flatten and sort
            values = sort(window(:));

            % Take median (5th element of 9)
            output(i, j) = values(5);
        end
    end
end
```

Co-design Bilateral filter script in MATLAB

bilateral_filter_matlab.m

```
function output = bilateral_filter_matlab(input)

    [h, w] = size(input);

    % Spatial weights
    spatial_weights = [4, 6, 4;
                       6, 9, 6;
                       4, 6, 4];

    % Add border padding
    padded = padarray(double(input), [1, 1], 'replicate');
    output = zeros(h, w, 'like', input);

    for i = 1:h
        for j = 1:w
            center = padded(i+1, j+1);
            window = padded(i:i+2, j:j+2);

            weighted_sum = 0;
            weight_sum = 0;

            % Process each pixel in window
            for m = 1:3
                for n = 1:3
                    spatial_w = spatial_weights(m, n);

                    % Range weight
                    intensity_diff = abs(center - window(m, n));
                    range_w = range_weight_lut_matlab(intensity_diff);

                    % Combined weight
                    bilateral_w = spatial_w * range_w;

                    weighted_sum = weighted_sum + window(m, n) * bilateral_w;
                    weight_sum = weight_sum + bilateral_w;
                end
            end

            if weight_sum > 0
                output(i, j) = weighted_sum / weight_sum;
            else
                output(i, j) = center;
            end
        end
    end
end

function range_w = range_weight_lut_matlab(intensity_diff)
% Range weight LUT
    if intensity_diff <= 0
        range_w = 255;
    elseif intensity_diff <= 1
        range_w = 253;
    elseif intensity_diff <= 2
        range_w = 248;
    elseif intensity_diff <= 3
        range_w = 240;
    elseif intensity_diff <= 4
        range_w = 230;
    elseif intensity_diff <= 5
        range_w = 218;
    elseif intensity_diff <= 10
        range_w = 165;
    elseif intensity_diff <= 15
        range_w = 118;
    elseif intensity_diff <= 20
        range_w = 82;
    elseif intensity_diff <= 25
        range_w = 55;
    elseif intensity_diff <= 30
        range_w = 36;
    elseif intensity_diff <= 40
        range_w = 15;
    elseif intensity_diff <= 50
        range_w = 6;
    else
        range_w = 0;
    end
end
```

Co-design Sobel filter script in MATLAB

sobel_filter_matlab.m

```
function output = sobel_filter_matlab(input)

    [h, w] = size(input);

    % Sobel kernels
    Gx = [-1, 0, 1;
          -2, 0, 2;
          -1, 0, 1];

    Gy = [-1, -2, -1;
           0,  0,  0;
           1,  2,  1];

    % Add border padding
    padded = padarray(double(input), [1, 1], 'replicate');
    output = zeros(h, w, 'like', input);

    for i = 1:h
        for j = 1:w
            % Extract 3x3 window
            window = padded(i:i+2, j:j+2);

            % Calculate gradients
            gx = sum(sum(window .* Gx));
            gy = sum(sum(window .* Gy));

            % Magnitude approximation |Gx| + |Gy|
            magnitude = abs(gx) + abs(gy);

            % Clamp to 0-255
            output(i, j) = min(max(magnitude, 0), 255);
        end
    end
end
```

# APPENDIX 2

| Formula | Description |
|---|---|
| $f(x) = \max(0,x)$ | Rectified Linear Unit (ReLu) function |
| $L_C = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$ | Cross-entropy loss |
| $L_Q = -\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ | Quadratic loss |
| $G(x,y) = \left(\frac{1}{16}\right)\sum_{i=-1}^{1}\sum_{j=-1}^{1}[I(x+i,y+j) * K(i+1,j+1)]$ | Gaussian Filtering |
| $M(x,y) = median\{I(x+i,y+j)\},\ \ for\ i,j \in [-1,0,1]$ | Median Filtering |
| $G = \lvert G_x\rvert + \lvert G_y\rvert$ | Gradient Magnitude for Sobel filter |
| $K_x = \begin{bmatrix}-1 & 0 & +1\\ -2 & 0 & +2\\ -1 & 0 & +1\end{bmatrix}$, $K_y = \begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ +1 & +2 & +1\end{bmatrix}$ | Operation Kernels of Sobel filter |
| $BF(x,y) = \frac{1}{W}\sum_{i=-1}^{1}\sum_{j=-1}^{1}[I(x+i,y+j) * w_s(i,j) * w_r(\lVert I(x+i,y+j) - I(x,y)\rVert)]$ | Bilateral Filtering |
| $\frac{Number\ of\ Correct\ Predictions}{Total\ Predictions}\ x\ 100\%$ | Training accuracy |
| $\frac{Number\ of\ Correct\ Predictions}{Total\ Predictions}\ x\ 100\%$ | Validation accuracy |
| $\frac{True\ Positives(c)}{True\ Positives(c) + False\ Positives(c)}$ x 100% | Precision |
| $\frac{True\ Positives(c)}{True\ Positives(c) + False\ Negatives(c)}$ x 100% | Recall |
| $\frac{2\ X\ Precision(c)\ X\ Recall(c)}{Precision(c) + Recall(c)}$ x 100% | F1-score |
| $\frac{Number\ of\ Correct\ Predictions}{Total\ Predictions}\ x\ 100\%$ | Testing Accuracy |

| | |
|---|---|
| $\frac{Total\ inference\ time}{Number\ of\ samples}$ | Inference Speed |
| $Improvement = \frac{M_{filter} - M_{benchmark}}{M_{benchmark}}$ x 100% | Relative Performance Improvement |
| $Resource\ Increase = \frac{R_{filter} - R_{benchmark}}{R_{benchmark}}$ x 100% | Resource Increase formula |

# AUTHOR'S PROFILE

Joel Laing Luyoh obtained his Diploma in Electrical Engineering (Electronic) in 2021 from Universiti Teknologi MARA, UiTM Samarahan Campus, Sarawak and received his Bachelor of Engineering (Hons.) Electrical and Electronic Engineering in 2024 from Universiti Teknologi MARA, UiTM Permatang Pauh Campus, Pulau Pinang. Then, he directly pursues his master's in electrical engineering at the same university he obtained his degree. His research focuses on machine learning approach, in particular convolutional neural network (CNN), for digit sign language classification as well as the implementation of field-programmable gate array (FPGA) platform. He believes that his education journey carves the clear direction towards the career he will pursue upon completing his masters.

## LIST OF PUBLICATIONS

Journal:

1. **J.L. Luyoh**, S. Setumin, E. Noorsal, S.J.A. Bakar, M.S. Hadi, "A systematic review of sign language classification system for real-time application" *J. Electrical annd Electronic Systems Research* ., 2025 (**MyCite**)
1. **J.L. Luyoh**, S. Setumin, E. Noorsal, S.J.A. Bakar, D.E. Cahyani, "Co-Design Filter Block for FPGA-CNN Sign Language Recognition: A Hardware-Software Ablation Study" *J. IAES International Journal of Artificial Intelligence (IJ-AI)* ., 2026 (Submitted – Under review) (**Scopus**)

Conference Paper:

2. **J.L. Luyoh**, S. Setumin, E. Noorsal, S.J.A. Bakar, "Comparative Analysis of Activation Functions in CNN for Digit Sign Language Classification" *J. Electrical annd Electronic Systems Research* ., 2025 (**Scopus**)

## Innovation Awards

1. **J.L. Luyoh**, S. Setumin, E. Noorsal, S.J.A. Bakar, D.E. Cahyani, "CNN-based Sign Language Recognition Core Architecture for ASIC on FPGA Prototype", *International Research and Information Science Expo (iRISE)*, 2026: ***Gold Medal (Professional Innovator)***
2. **J.L. Luyoh**, S. Setumin, E. Noorsal, S.J.A. Bakar, D.E. Cahyani, "CNN-based Sign Language Recognition Core Architecture for ASIC on FPGA Prototype", *International Research and Information Science Expo (iRISE)*, 2026: ***Special Award Digital Inclusion Laureate***.

## Copyright

**J.L. Luyoh**, S. Setumin, E. Noorsal, S.J.A. Bakar, D.E. Cahyani, "CNN-based Sign Language Recognition Core Architecture for ASIC on FPGA Prototype",
Copyright No: CRLY2026P00143

**List of Awards and Certificates :**

1. 15th International Conference on Control System, Computing & Engineering (ICCSCE) 2025, ***Participation Presentation Certificate***

Session 4A
ID1048

IEEE INTERNATIONAL CONFERENCE ON CONTROL SYSTEM, COMPUTING & ENGINEERING
15th ICCSCE 2025 PENANG, MALAYSIA

CERTIFICATE

It is hereby certified that

**Joel Laing Luyoh**

has presented a paper

**Comparative Analysis of Activation Functions in CNN for Digit Sign Language Classification**

*Joel Laing Luyoh, Samsul Setumin, Emilia Noorsal, Siti Juliana Abu Bakar*

in conjunction with

15th IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2025)

Jointly organized by

IEEE Control Systems Society, Malaysia Section Chapter and
School of Electrical Engineering, College of Engineering, Universiti Teknologi MARA (UiTM)

on 22 - 23 August 2025

**Associate Prof. Dr. Hjh. Siti Noraini Sulaiman**
**General Chair ICCSCE2025**

IEEE CHAPTER OF CONTROL SYSTEM SOCIETY MALAYSIA SECTION
UNIVERSITI TEKNOLOGI MARA

2. 15th International Conference on Control System, Computing & Engineering (ICCSCE) 2025, ***Best Paper Award Certificate***

IEEE INTERNATIONAL CONFERENCE ON CONTROL SYSTEM, COMPUTING & ENGINEERING
15th ICCSCE 2025 PENANG, MALAYSIA

CERTIFICATE

This certificate is proudly presented to

**Joel Laing Luyoh, Samsul Setumin, Emilia Noorsal Siti and Juliana Abu Bakar**

In recognition of being awarded for **Best Paper** entitled

**Comparative Analysis of Activation Functions in CNN for Digit Sign Language Classification**

in conjunction with

15th IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2025)

Jointly organized by

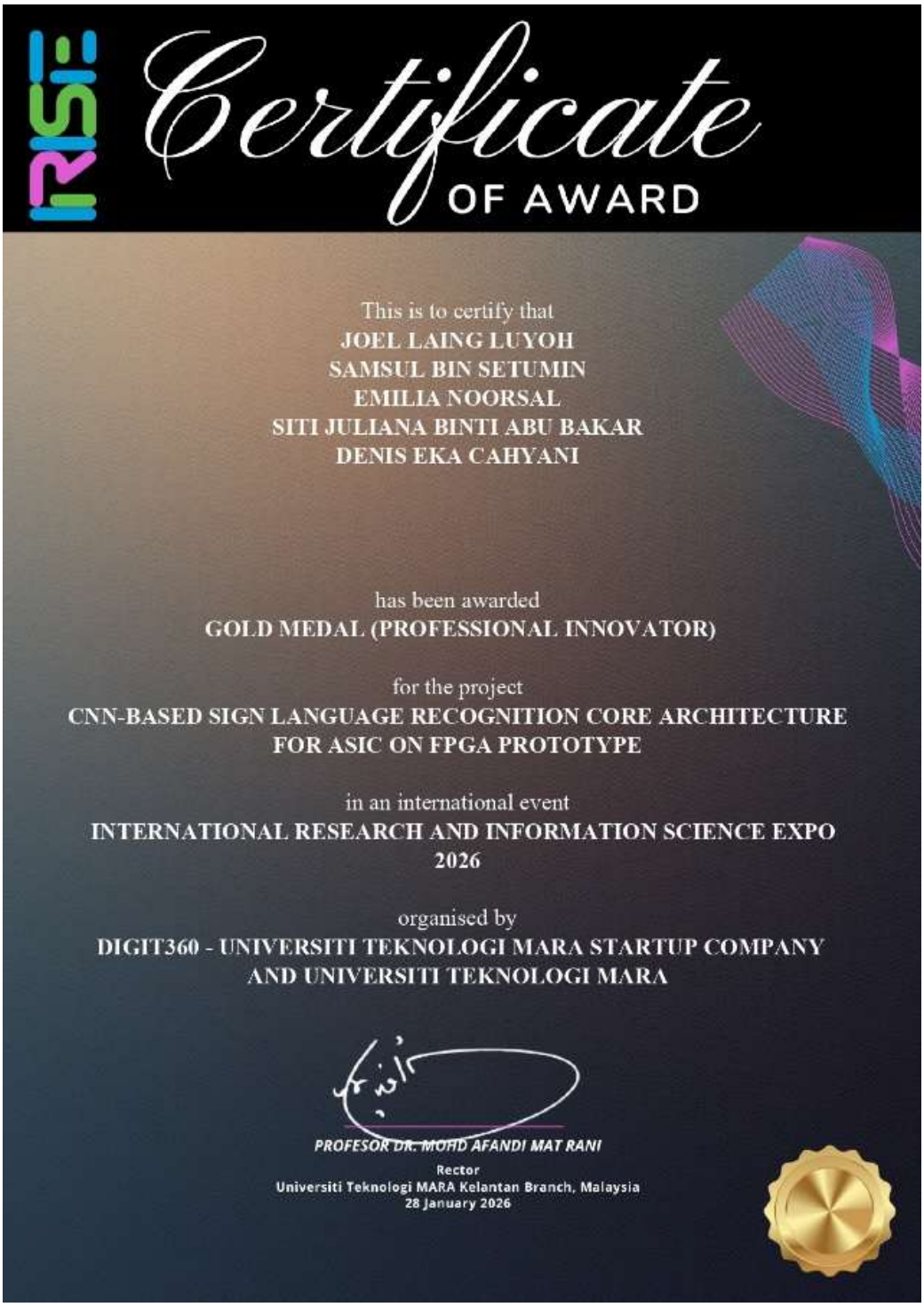
IEEE Control Systems Society, Malaysia Section Chapter and
School of Electrical Engineering, College of Engineering, Universiti Teknologi MARA (UiTM)

on 22 - 23 August 2025

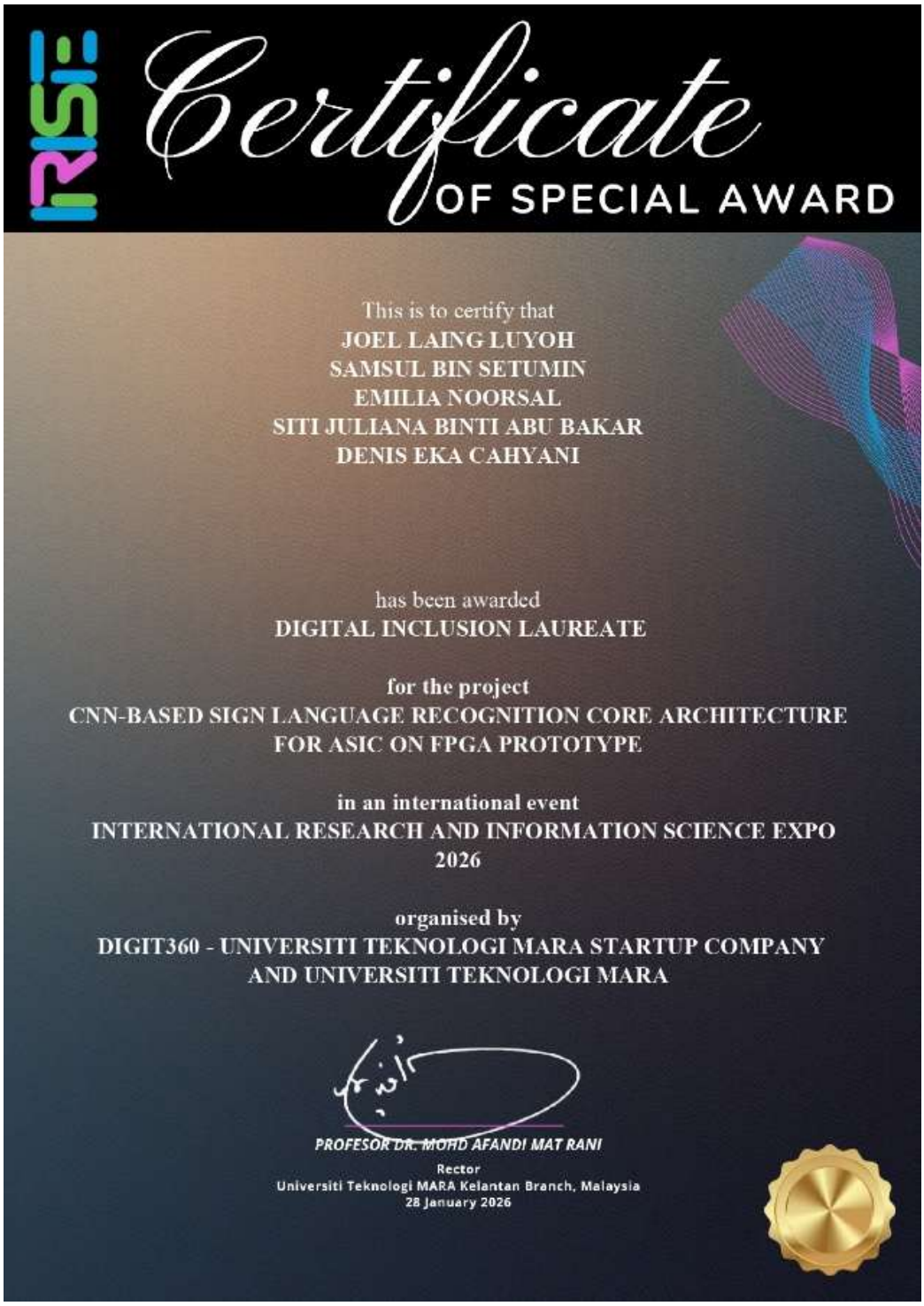
**Associate Prof. Dr. Hjh. Siti Noraini Sulaiman**
**General Chair ICCSCE2025**

IEEE CHAPTER OF CONTROL SYSTEM SOCIETY MALAYSIA SECTION
UNIVERSITI TEKNOLOGI MARA

3. International Research and Information Science Expo (iRISE) 2026, ***Gold Medal Award (Professional Innovator) Certificate***



4. International Research and Information Science Expo (iRISE) 2026, ***Digital Inclusion Laureate Award Certificate***



5. Literary Copyright on title of work : CNN-based sign language recognition core architecture for ASIC on FPGA prototype, ***Certificate of Notification of Copyright - CRLY2026P00143***

**AKTA HAK CIPTA 1987 I** *COPYRIGHT ACT 1987*
**PERATURAN-PERATURAN HAK CIPTA (PEMBERITAHUAN SUKARELA) 2012**
*COPYRIGHT (VOLUNTARY NOTIFICATION) REGULATIONS 2012*

**Sijil Pemberitahuan** *Hak Cipta*

*Certificate of Notification of Copyright*

Nombor Pemberitahuan
*(Notification Number)*

**CRLY2026P00143**

*Sijil Pemberitahuan ini dikeluarkan menurut seksyen 26B Akta Hak Cipta 1987 [Akta 332]. Hak cipta dalam karya telah dimaklumkan kepada Pengawal pada tarikh pemberitahuan dan seperti yang diperincikan di sini.*

*This Certificate of Notification is issued pursuant to section 26B of the Copyright Act 1987 [Act 332]. The copyright in the work was notified to the Controller on the date of notification and as detailed herein.*

**12 JANUARI 2026**
Tarikh Pemberitahuan
*(Date of Notification)*

Tajuk Karya *(Title of Work)*
**CNN-BASED SIGN LANGUAGE RECOGNITION CORE ARCHITECTURE FOR ASIC ON FPGA PROTOTYPE**

Kategori Karya *(Category of Work)*
**SASTERA** *(LITERARY)*

Tarikh Karya Dicipta *(Date of Creation)*
**12 JANUARI 2026**

Tarikh & Negara Penerbitan Pertama
*(Date & Country of First Published)*
**TIDAK BERKAITAN**

Pemunya *(Owner)*
**UNIVERSITI TEKNOLOGI MARA CAWANGAN PULAU PINANG, 13500 PERMATANG PAUH PULAU PINANG, MALAYSIA**

Pencipta *(Author)*
**JOEL LAING LUYOH (001007130241), SAMSUL BIN SETUMIN (831019015825), EMILIA BINTI NOORSAL (740403065188), SITI JULIANA BINTI ABU BAKAR (851224145192), DENIS EKA CAHYANI (E4713399)**

**YUSNIEZA SYARMILA BINTI YUSOFF**
**PENGAWAL HAK CIPTA MALAYSIA**
*COPYRIGHT CONTROLLER OF MALAYSIA*

PENGAWAL HAK CIPTA MALAYSIA

Page 1 of 1