MALMÖ UNIVERSITY

Letter Recognition for Handwriting on Embedded System Using a Machine Learning Model

Mohammad Abdulsalam Hajjo
Ba Duc Tang

Supervisor:
Mahtab Jamali

Examiner:
Zahra Asghari Varzaneh

Degree: Bachelor of Science in Engineering, 180 hp
Main field: Computer Science
Program: Bachelor of Science in Engineering in Computer Science
Date for examination: 2025-06-03

Abstract

This thesis investigates the development and deployment of a lightweight machine learning model for real-time handwritten letter recognition on a highly resource-constrained embedded system, the ESP32-CAM microcontroller. The project explores two convolutional neural network (CNN) architectures: a quantized version of the pre-trained MobileNetV2 and a custom-designed SimpleCNN. Both models were adapted for efficient inference within the ESP32-CAM's strict memory and processing limits.

The system uses the EMNIST Letters dataset, with preprocessing and data augmentation to enhance model generalization across varying handwriting styles. The MobileNetV2 model achieved an F1 score of 87.4% with an inference time of 294 ms, while the SimpleCNN model reached 92.8% accuracy in a simulated environment before hardware deployment. On the other hand, the deployed model only utilizes around 20% of the ESP32-CAM's available RAM... there is potential to use a model up to five times larger, which could improve recognition performance.

While the system shows that accurate handwritten letter recognition is feasible on embedded platforms, its scope is limited to uppercase letters and balanced datasets due to platform and training constraints. The results underscore the potential for using embedded deep learning in portable, assistive, and educational devices, provided that limitations in model size, training time, and dataset complexity are carefully managed.

Keywords: Handwriting recognition, embedded systems, ESP32-CAM, convolutional neural networks (CNN), MobileNetV2, EMNIST dataset, model quantization, real-time inference, machine learning.

Sammanfattning

Denna uppsats undersöker utvecklingen och implementeringen av en lättviktig maskininlärningsmodell för realtidsigenkänning av handskrivna bokstäver på ett resursbegränsat inbyggt system, mikrokontrollern ESP32-CAM. Projektet utforskar två konvolutionella nätverksarkitekturer (CNN): en kvantiserad version av den förtränade MobileNetV2 och en egenutvecklad modell, SimpleCNN. Båda modellerna anpassades för effektiv inferens inom ESP32-CAM:s begränsade minnes- och beräkningskapacitet.

Systemet använder EMNIST Letters-datasetet, med förbehandling och data-augmentation för att förbättra modellens generalisering över olika handstilar. MobileNetV2-modellen uppnådde ett F1-värde på 87,4 % med en inferenstid på 190 ms, medan SimpleCNN-modellen nådde en noggrannhet på 92,8 % i en simulerad miljö innan den implementerades på hårdvaran. Å andra sidan använder den utplacerade modellen bara cirka 20 % av ESP32-CAM:s tillgängliga RAM, det finns potential att använda en modell upp till fem gånger större, vilket kan förbättra igenkänningsprestandan.

Även om systemet visar att exakt igenkänning av handskrivna bokstäver är genomförbart på inbyggda plattformar, är omfattningen begränsad till versaler och balanserade dataset på grund av plattforms- och träningsbegränsningar. Resultaten betonar potentialen för att använda inbyggd djupinlärning i bärbara, assistiva och pedagogiska enheter, förutsatt att begränsningar i modellstorlek, träningstid och datakomplexitet hanteras noggrant.

Nyckelord: Handskriftsigenkänning, inbyggda system, ESP32-CAM, konvolutionella neurala nätverk (CNN), MobileNetV2, EMNIST-dataset, modellkvantisering, realtidsinferens, maskininlärning.

TABLE OF CONTENTS

1. Introduction

1.1 Background

Handwriting recognition has long been a complex problem in computer vision and learning due to the highly variable nature of human writing [1]. Differences in letter shapes, sizes, slants, and writing speeds can make accurate recognition difficult for machines. Although humans can easily interpret a wide range of handwriting styles, the task has historically been very challenging and complicated for computers.

With the development of machine learning, it has become significantly more feasible for machines to process and interpret human handwriting. Machine learning algorithms such as Convolutional Neural Networks (CNN) have made it possible for machines to recognize human-written text with remarkably high accuracy [2]. Despite their high accuracy, CNN-based handwriting recognition models face several limitations. Firstly, they are highly sensitive to input variations, such as inconsistent writing styles, skewed or noisy characters, and differences in writing instruments or paper quality. Secondly, CNNs require large, annotated datasets for effective training, which may be difficult to obtain for some languages or user groups. In addition, these models often rely on careful preprocessing to ensure accurate character segmentation and alignment, which adds to the system's complexity. Lastly, there is a critical trade-off between model accuracy and inference speed. Deeper architectures, such as ResNet or EfficientNet, offer better accuracy but often incur high computational costs, making them less suitable for real-time applications or deployment on resource-constrained devices.

These advancements open the door to a wide range of real-world applications. For instance, handwriting recognition systems can support individuals with dyslexia, motor impairments, or vision challenges by converting handwritten content into digital formats, enhancing accessibility and comprehension [3][4]. In healthcare settings, such systems can help patients read doctors' handwritten prescriptions more accurately, reducing the risk of medication errors due to illegible handwriting, which is a common issue in clinical environments [5].

CNN-based handwriting recognition can also automate postal mail sorting and enhance document management systems, particularly in cultural heritage preservation [5][6]. These systems are currently being tested in practical applications like note-taking assistance for students and AI-supported tutoring, where the system can recognize handwritten input and provide real-time feedback [5].

To enable such practical applications, CNN-based handwriting recognition models have become more accurate and reliable by improving their ability to handle variations in handwriting style, background noise, and distortion [5]. To address these variations, techniques like data augmentation and optimized convolutional architectures help the models generalize better across

different inputs. CNNs are sometimes integrated with attention mechanisms or sequential models (such as RNNs or LSTMs) to better handle temporal dependencies and character continuity [6].

While handwriting recognition models have achieved high accuracy when deployed on systems with sufficient computational resources, realizing the same functionality on embedded systems remains a major challenge [1]. Microcontrollers, which are commonly used in embedded systems, have limited memory, processing power, and storage capacity. These constraints make it difficult to deploy large machine learning models [7,8].

Embedded systems are typically small and have limited computational capacity, which demands careful optimization of the deployed model. Due to their constrained hardware, deploying a standard neural network model onto an embedded device is not feasible without significant modification [9]. To make deployment possible, models must be carefully optimized through techniques such as quantization, which reduces the precision of numerical values to lower memory and energy demands, and pruning, which removes less important parameters to shrink the model's size. It may also involve tailoring the model to fit within the device's internal memory and minimizing the number of computations required [10]. These optimization techniques are vital to ensuring that models can run in real-time on devices with tight energy budgets. For instance, quantized models operate with 8-bit integers instead of 32-bit floating points, which drastically reduces memory usage and speeds up inference without significantly sacrificing accuracy. Pruning, on the other hand, eliminates redundant weights, resulting in a sparser and more efficient network. Furthermore, techniques like model distillation and architecture search (e.g., using MobileNet variants) can be applied to create compact models suited for resource-limited environments.

1.2 Purpose and Goal

This study investigates the feasibility of implementing handwriting recognition directly on a microcontroller by designing, optimizing, and testing a convolutional neural network tailored to hardware constraints.

The primary purpose of this research is to design and implement an embedded system capable of recognizing handwritten letters with high accuracy. This system aims to operate independently on the ESP32-CAM without relying on external computing resources, thereby enabling efficient, low-power, and real-time inference.

The central goal is to explore how machine learning models can be modified and optimized to perform effectively under hardware limitations. Key performance aspects such as inference speed, memory usage, and classification accuracy are assessed to determine the practical viability of deploying handwriting recognition systems on low-power embedded devices for real-world applications such as assistive technology, education, or portable note-taking tools.

1.3 Research questions

- **RQ1:** What approaches can be used to train a machine learning model that recognizes handwritten letters while remaining efficient enough for microcontroller deployment?

- **RQ2:** What dataset types and preprocessing techniques are effective in managing the wide variation in handwriting styles?

- **RQ3:** How can the model be adapted to run on a microcontroller with limited resources, such as memory, processing power, and camera?

1.4 Delimitations

This study focuses on deploying a CNN on the ESP32-CAM microcontroller, which is limited in both memory and processing power. The work emphasizes adapting existing models to run efficiently on this platform rather than developing complex or large-scale architectures. Inference is performed entirely on the device, without support from external hardware or cloud-based services. The dataset was restricted to uppercase letters from the EMNIST set, which may limit the model's ability to generalize to broader handwriting styles or mixed-case input. Testing was conducted under controlled conditions, primarily using short-term performance metrics such as inference time, memory usage, and model size. Real-world robustness, user variability, and long-term deployment were not part of this evaluation.

2. Theoretical Background

2.1 Artificial Intelligence (AI)

Artificial Intelligence refers to the ability of digital computers or computer-controlled machines to perform tasks commonly associated with human intelligence [11]. These tasks include reasoning, problem solving, learning from experience, understanding language, and recognizing patterns or objects. AI systems are designed to mimic aspects of human cognitive function and are found in a variety of applications such as medical diagnostics, handwriting recognition, and speech processing. While current AI models can achieve expert-level performance in narrowly defined tasks, they still lack the broad adaptability and general knowledge characteristic of human intelligence.

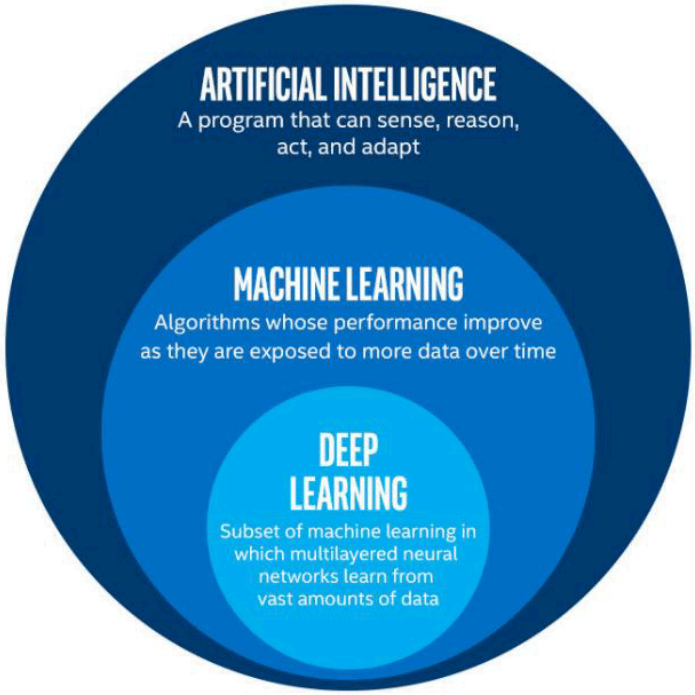

Figure 1: AI, ML, and DL [34]

2.2 Machine Learning (ML)

Machine learning, a subfield of artificial intelligence (AI), enables systems to learn from data and improve their performance over time without being explicitly programmed for every specific task. It involves developing algorithms that can identify patterns in data, make decisions, or predict outcomes based on learned experience. ML techniques are typically categorized into supervised learning, unsupervised learning, and reinforcement learning, each with distinct training strategies and applications [12].

In the context of handwriting recognition, machine learning allows models to generalize across different writing styles by training on labeled image datasets and refining predictions through iterative learning. During training, a model is exposed to large amounts of input data and adjusts its internal parameters to minimize prediction errors. Among the various learning approaches, supervised learning is most commonly used in handwriting recognition. In this method, the system is trained on labeled images of handwritten letters, learning to associate specific pixel patterns with corresponding letters or digits.

2.3 Deep Learning (DL)

Deep learning is a subset of machine learning that employs multilayered neural networks, known as deep neural networks, to simulate the complex decision-making capabilities of the human brain [13]. These networks consist of multiple layers of interconnected nodes, enabling the system to learn from vast amounts of unstructured data and automatically extract features and representations. This hierarchical learning approach allows deep learning models to excel in tasks such as image and speech recognition, natural language processing, and autonomous systems. Unlike traditional machine learning models that often require manual feature extraction, deep learning models can learn directly from raw data, making them highly effective for complex problem-solving

CNNs are particularly well-suited for handwriting recognition because they can automatically extract local features such as curves, strokes, and edges that are critical for distinguishing characters. However, deploying CNNs on a low-power microcontroller like the ESP32-CAM introduces unique challenges, including limited memory, restricted processing capacity, and energy constraints. To address these limitations, lightweight architectures and optimization strategies such as quantization, pruning, and model compression can be explored.

2.4 Convolutional Neural Networks (CNN)

CNN is a type of deep learning model that is suited to analyzing visual data. They are particularly effective for tasks like image classification, object detection, and handwriting recognition due to their ability to automatically learn spatial hierarchies of features from raw pixel data [14].

A typical CNN consists of multiple layers, including convolutional layers, activation functions, pooling layers, and fully connected layers [14]. Convolutional layers (kernels) apply filters that slide over the input image to detect local features such as edges, curves, or textures. These filters allow the network to extract low-level patterns that are critical in the early stages of visual analysis. After each convolution, a non-linear activation function like ReLU (Rectified Linear Unit) is applied to introduce non-linearity, allowing the model to learn complex mappings. Pooling layers are used to downsample the data by reducing its spatial dimensions while preserving important features. This reduces computational cost and helps the network become more robust to small shifts or distortions in the input. Finally, the output is passed to one or more

fully connected layers, where all neurons are connected and the network makes its final prediction based on the learned features.

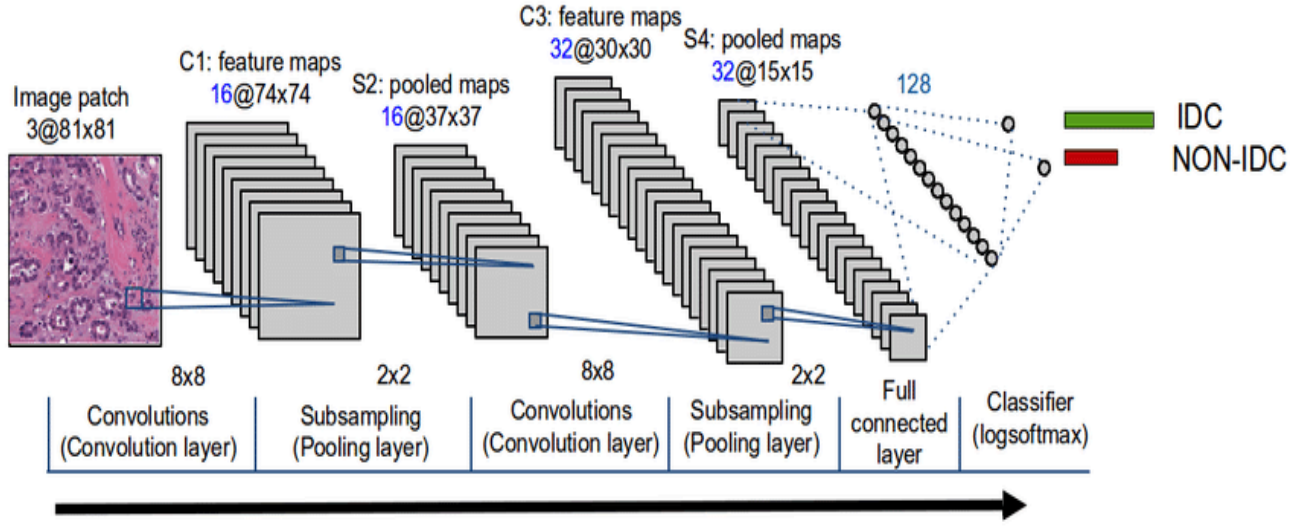

Figure 2: Typical layers of CNN [35]

In handwriting recognition, CNNs are effective because they can recognize the unique shapes and strokes of individual characters. As the network trains on labeled images of handwritten letters, it progressively learns to identify the visual patterns that distinguish one character from another, enabling accurate classification.

2.5 Embedded system

Embedded systems are specialized computing systems that combine hardware and software to perform a dedicated task or set of tasks, often within a larger mechanical or electrical system. Embedded systems are built to execute specific functions efficiently, reliably, and often in real-time. They are commonly found in everyday devices such as washing machines, traffic lights, smartphones, medical equipment, and industrial machinery [15].

A typical embedded system consists of a microcontroller or microprocessor, memory, I/O interfaces, and dedicated software stored in firmware. These components are tightly integrated and tailored for minimal power consumption, compact physical design, and efficient performance. Embedded systems may run continuously without human intervention and are often designed to operate under strict performance and reliability constraints [15].

The ESP32-CAM offers built-in Wi-Fi, Bluetooth capabilities, and a dual-core processor, making it a cost-effective and energy-efficient solution for real-time applications. However, deploying machine learning models, particularly convolutional neural networks, on such a device presents a significant challenge due to its limited RAM, modest clock speed, and restricted storage

2.6 Dataset

In the context of machine learning, a dataset is defined as a structured collection of data used to train, validate, and test models. In handwriting recognition, such datasets typically consist of labeled grayscale images, each representing a handwritten character or digit. These labeled examples enable the model to learn spatial and visual patterns associated with different characters, which is essential for achieving accurate classification results in character recognition tasks [16].

The Extended MNIST (EMNIST) dataset is a widely used benchmark in handwritten character recognition tasks. Introduced by Gregory Cohen et al. (2017), EMNIST extends the original MNIST dataset, incorporating both digits and letters, offering a more comprehensive and diverse dataset for training, validation, and testing machine learning models. These datasets contain images, each representing a handwritten character or digit. Such structured datasets are essential for enabling models to learn spatial and visual patterns critical for accurate classification in handwriting recognition tasks [16].

EMNIST is divided into several subsets, each tailored for specific research needs:

- **EMNIST ByClass**: Comprises 814255 samples across 62 unbalanced classes, distinguishing between digits (0-9), uppercase, and lowercase letters.
- **EMNIST ByMerge**: Also includes 814255 samples but merges similar uppercase and lowercase letters, resulting in 47 unbalanced classes.
- **EMNIST Balanced**: A subset with 131600 samples evenly distributed across 47 classes, useful for evaluating model performance under balanced conditions.
- **EMNIST Digits**: Contains 280000 samples across 10 balanced digit classes, serving as an extended version of the original MNIST digits.
- **EMNIST MNIST**: Replicates the original MNIST dataset configuration with 70000 digit samples in 10 balanced classes, ensuring compatibility with existing MNIST-based models.
- **EMNIST Letters**: Consists of 145600 images uniformly distributed across 26 balanced classes, representing uppercase English letters (A-Z).

Using the EMNIST letters dataset offers significant advantages. Its balanced class distribution and extensive size make it ideal for training robust handwriting recognition models that generalize well across diverse handwriting styles. This diversity is crucial for real-world applications where handwriting can vary significantly between individuals. Previous research has shown that convolutional neural networks (CNNs) trained on EMNIST perform effectively, particularly when combined with data augmentation and regularization techniques [17].

2.7 Feature extraction

Feature extraction is a key step in the machine learning pipeline where raw data, such as images of handwritten letters, is transformed into a structured set of numerical features that represent the most informative patterns in the input. This process simplifies the data and emphasizes critical

characteristics while reducing irrelevant information, making it more suitable for classification or prediction tasks [18].

In the context of handwriting recognition, feature extraction involves identifying visual cues such as edges, curves, corners, loops, and strokes that distinguish one character from another. Convolutional Neural Networks (CNNs) perform this process automatically through their convolutional layers, which apply multiple learnable filters across the input image. The initial layers detect low-level features like edges and gradients, while deeper layers capture higher-level patterns such as shapes or even entire character structures.

For CNN-based features, the number and size of convolutional filters must be carefully selected to balance between expressive power and computational cost. Larger feature maps and deeper networks improve accuracy but increase memory usage and inference time. To address this, lightweight CNN architectures are designed to extract discriminative features efficiently without exceeding the device's limited memory and processing capabilities.

2.8 Classification

Classification is a type of supervised machine learning task where a model is trained to assign input data to one of several predefined categories or classes. The algorithm learns to identify relationships and patterns in a labeled dataset, where each input is associated with a known output. Once trained, the model can predict the correct class for new, unseen data based on what it has learned during training [19].

There are two main types of classification: binary classification, where data is divided into two categories (e.g, spam vs not spam), and multi-class classification, where the model chooses from more than two classes. In the case of handwriting recognition, the task is a multi-class classification problem, where each letter represents a unique class. CNNs are well-suited for classification tasks involving visual data. A CNN processes an input image and extracts features such as edges, curves, and textures. These features are then passed through fully connected layers to determine which character class the image most likely represents [19].

2.9 Data augmentation

Data augmentation is a technique used in machine learning to increase the size and variability of training datasets by generating modified versions of existing data. This process improves model generalization and robustness, especially when the dataset is imbalanced [20].

Typical image augmentation methods include geometric transformations such as rotation, flipping, scaling, cropping, and random erasing, as well as photometric transformations like brightness adjustment, saturation change, and grayscale conversion. Some techniques, such as noise injection, add random pixel disturbances to help models become more resistant to visual distortions. These operations are often applied during training to enhance the model's ability to recognize patterns under varied conditions [20]

In this project, data augmentation is performed using automated tools provided by Edge Impulse, which apply a set of built-in transformations during training. Data augmentation also plays a crucial role in compensating for the limited size of the handwritten letter dataset. Several augmentation strategies were applied during training, including random rotations, horizontal flips, and other photometric adjustments, to increase variation and enhance recognition across different handwriting styles. This is especially important when deploying the model on the ESP32-CAM microcontroller, where the compact model architecture limits representational capacity. By exposing the model to a wide range of simulated input conditions during training, augmentation improves inference accuracy under real-world usage while keeping the model lightweight and efficient.

2.10 Confusion matrix

A confusion matrix is a performance measurement tool used to evaluate the results of a classification model. It organizes predictions into a grid that shows how many instances were correctly or incorrectly classified for each class, making it especially useful for diagnosing model behavior in both binary and multiclass classification tasks [21].

It presents the number of correct and incorrect predictions broken down by each class, typically organized into 4 categories:

- **True Positives (TP)**: Correctly predicted positive instances
- **True Negatives (TN)**: Correctly predicted negative instances
- **False Positives (FP)**: Incorrectly predicted positives
- **False Negatives (FN)**: Incorrectly predicted negatives

The key performance metrics are calculated as follows:

- **Precision** $= \frac{TP}{TP + FP}$ **(1)**
- **Recall** $= \frac{TP}{TP + FN}$ **(2)**
- **F1 Score** $= 2 \bullet \frac{Precision \bullet Recall}{Precision + Recall}$ **(3)**

This structure provides the foundation for calculating several key performance metrics. Precision measures how many predicted positives were correct, and Recall indicates how many positive cases the model successfully identified. The F1 score, which is the harmonic mean of precision and recall, is particularly valuable when evaluating models on imbalanced datasets where one class appears more frequently than others [21]

3. Related Work

3.1 Human Detection with A Feedforward Neural Network for Small Microcontrollers

In this article, L. Wulfert et al [21] address the problem domain of smart home applications, specifically focusing on reliable human detection for motion-controlled systems. The main problem they tackle is distinguishing human movements from non-human motions, for example, those caused by cars or animals using microcontrollers.

The authors propose a research approach combining unique image pre-processing techniques with a simple feedforward artificial neural network (ANN). This innovative method significantly reduces the ANN parameter count by over 99% compared to traditional deep neural networks (DNN) methods. This enables the use of low-power and low-storage microcontrollers, specifically ESP32.

Previous solutions typically employed advanced neural network methods like convolutional neural networks (CNNs) or region-based CNN (R-CNN). But on the other hand, these approaches required powerful hardware with large storage and were therefore unsuitable for low-performance embedded systems. Alternative research approaches, for example, using TensorFlow Lite for microcontrollers, resulted in prohibitively slow detection speeds unsuitable for real-time applications. [21]

The proposed method significantly reduces neural network parameters by more than 99% compared to conventional CNN approaches, achieving high accuracy with an F1-score of approximately 85% and real-time detection capabilities at 83 ms per frame on an ESP32.

Comment: While their work targets motion detection, it demonstrates that neural models can be adapted for ESP32 with reduced complexity.

3.2 Human Activity Recognition using Wireless Signals and Low-Cost Embedded Devices

In 2024, T. V. A. Tong, B. Bui-Thanh, and P. N. T. H [22] investigated human activity recognition (HAR) for smart home applications, emphasizing affordability and user privacy. The main problem addressed is the reliance on expensive or privacy-invasive techniques, such as wearable sensors and camera-based monitoring. Their research proposes using channel state information (CSI) extracted from wifi signals as a non-intrusive and cost-effective alternative.

They utilize low-cost ESP32 microcontrollers for CSI data collection, combined with machine learning algorithms deployed on a Jetson Nano for activity classification.

In contrast to previous studies employing expensive and complex hardware like Intel network interface cards (NICs) or universal software radio peripheral (USRP) devices. This work leverages and enhances the practicality of Steven M. Hernandez's ESP32 CSI toolkit. Their experiments show promising results, achieving high accuracy, around 95,57%, and minimal latency. Thus validating the ESP32's effectiveness and scalability for real-time, privacy-aware HAR in smart home environments. [22]

Comment: Although the sensing method differs, this study also confirms the viability of ESP32 for real-time machine learning.

3.3 Real-time Handwritten Digit Recognition Using CNN on Embedded Systems

Published in 2024, the study by I. Mchichou et al [23] focused on implementing a Convolutional Neural Network (CNN) model trained with deep learning on an embedded system. It explored a digit recognition system on a Raspberry Pi 3 platform. The authors developed a novel CNN architecture consisting of two convolutional layers and three fully connected layers. The model was trained on the MNIST dataset, which comprises a training set of 60,000 images and a test set of 10,000 images. Their approach achieved an impressive accuracy of 99,94%.

The equipment used by the authors includes a Raspberry Pi 3 Model B+ board equipped with a quad-core ARM Cortex-A53 processor, 1 GB of RAM, a 32 GB SD card for data storage, a 5MP Pi Camera, and an LCD screen. The camera was utilized to capture handwritten digits in real-time, and the LCD screen simply displays the results. The system demonstrated efficient performance, showcasing the feasibility of implementing CNN on embedded systems.

Comment: Their system achieved high accuracy, though it was implemented on more powerful hardware (Raspberry Pi), and the system is only able to recognize digits.

3.4 Real-Time Handwritten Letters Recognition on an Embedded Computer Using ConvNets

In 2018, D. Nunez and S. Hosseini [17] published a study on real-time handwritten letters recognition on an embedded system. The authors introduced a lightweight CNN model specifically designed for recognizing 26 handwritten letters, both uppercase and lowercase, on a Raspberry Pi 3 platform. They trained the model with the EMNIST dataset, which has a total of 145,600 uppercase/lowercase letters divided into 26 balanced classes. The authors allocated 86% of these images for training and 14% for testing, meaning 124,800 images were used for training and 20,800 images for testing. Their approach achieved an accuracy of 93.4% with a response time of 21.9 ms, making the system nearly instantaneous.

The CNN model the authors designed consisted of two convolutional layers with 5x5 kernels, followed by max-pooling layers and two fully connected layers with 500 neurons each. The model was specifically designed to be able to work with low computational resources, making it suitable for embedded systems . The authors used Caffe for training and OpenCV for real-time inference on the Raspberry Pi 3.

Comment: This work aligns closely with the goals of this thesis, but is limited to the Raspberry Pi.

3.5 Handwritten English Character and Digit Recognition

In 2021, Al-Mahmud et al. [25] conducted a study on handwritten character and digit recognition using CNNs to classify English letters and digits. Their primary goal was to enhance the accuracy of a previously developed CNN architecture by tuning hyperparameters and applying dropout regularization to mitigate overfitting. The study began with the MNIST dataset, achieving a test accuracy of 99.47%, and was later expanded to include a custom merged dataset consisting of both the MNIST digits and the A-Z handwritten dataset. The dataset includes 442,450 images spanning 36 classes, and the model achieved 98.94% accuracy.

The CNN architecture included two convolutional layers with ReLU activations and pooling layers for downsampling, followed by flattening and fully connected layers. Dropout was applied after dense layers to improve generalization. The architecture was modified slightly depending on whether it was trained on MNIST alone or the combined dataset. The experiments showed that adjusting dropout rates and learning rates helped reduce overfitting and improve performance.

Comment: The author's focus was on improving accuracy through regularization on general-purpose hardware.

3.6 Handwriting Recognition using Convolutional Neural Network and Support Vector Machine Algorithms

Latchoumy et al. [26] proposed a hybrid handwriting recognition system that integrates CNN and Support Vector Machine (SVM) to recognize both handwritten English letters and digits using the EMNIST dataset.

The hybrid model employed CNN for alphabet recognition due to its ability to automatically extract hierarchical features from image data, and SVM for digit recognition based on its strong performance in binary classification tasks. This dual-approach aimed to leverage the strengths of Both algorithms to maximize classification performance across diverse input types.

The EMNIST dataset was preprocessed by converting RGB images to grayscale and standardizing to 28×28 pixels. CNN was applied with multiple convolutional layers, ReLU activations, and a final softmax output layer, while the SVM used a hyperplane-based approach to separate digit classes. Feature extraction was further supported by the use of the Grey Level Co-occurrence Matrix to characterize texture in the input images.

The model achieved an overall accuracy of 95.41%, with CNN reaching 95.95% on character recognition and SVM achieving 93.95% on digits.

Comment: While their hybrid model aimed to optimize accuracy, it was not designed for constrained environments.

Summary

Previous research has demonstrated the effectiveness of CNN-based handwriting recognition systems; however, most implementations rely on relatively powerful embedded platforms or focus solely on digit recognition. Additionally, several studies prioritize model accuracy without addressing hardware constraints, or they employ hybrid approaches that are impractical for real-time inference on ultra-low-power devices. A notable gap remains in demonstrating full letter recognition on extremely resource-limited microcontrollers like the ESP32.

4. Method

The Systems Development Research Methodology (SDRM), introduced by Nunamaker et al [27], was selected for this study because it offers a comprehensive framework for both constructing and evaluating technical systems. The project involves developing a handwriting recognition system on a constrained embedded device (ESP32-CAM), which requires the integration of theoretical understanding and practical implementation. SDRM is well aligned with this objective, as it emphasizes both conceptual analysis and iterative system development, allowing for systematic exploration of design decisions under real-world limitations.

The methodology begins by establishing a conceptual foundation through a review of prior research on embedded neural networks, which helps define the scope and feasibility of the system. A modular architecture is then designed, incorporating components for image capture, model inference, and display output. The choice of CNN models, MobileNetV2 and a custom SimpleCNN, reflects contrasting priorities between accuracy and resource efficiency, enabling comparative evaluation within the development process. Each model is trained, quantized, and deployed on the ESP32-CAM, demonstrating compatibility with the device's memory and processing constraints.

SDRM further supports structured system testing, including assessment of classification accuracy, inference time, and memory usage. The iterative nature of the framework enables refinement of both hardware and software components across development cycles. This methodology ensures that decisions at each stage are grounded in both empirical results and practical system constraints, making it particularly suitable for research involving embedded machine learning applications.

The SDRM framework is implemented through two iterative modules - Iteration 1 and Iteration 2- each reflecting a different CNN model used during the development process.

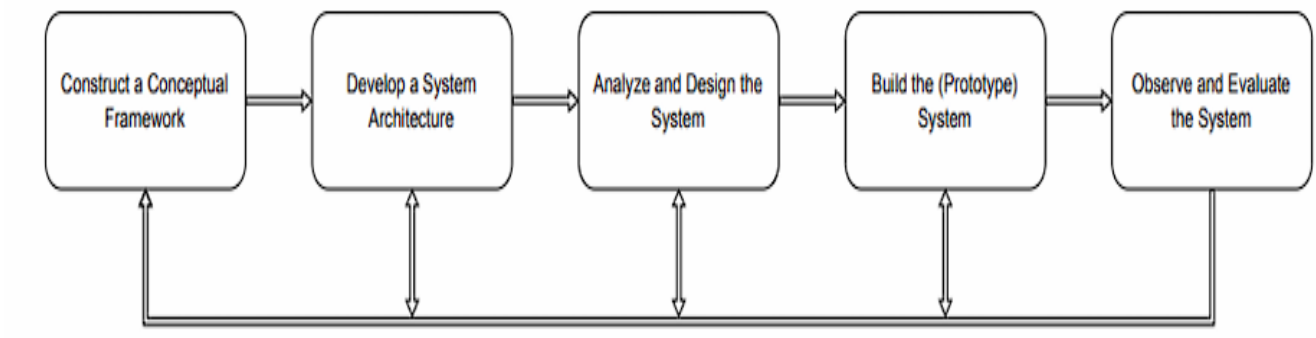

Figure 3: Research process in SDRM [36]

4.1 Overview of the Method

The SDRM includes five key phases:

- **Construct a Conceptual Framework**: A literature review and environmental analysis were conducted to assess the feasibility of deploying machine learning models on constrained embedded hardware. This helped establish the research scope and constraints of the system.

- **Develop a System Architecture**: A modular system architecture was designed with components for image acquisition (camera), processing (CNN inference), and output (LCD display). The ESP32-CAM was selected due to its low cost, integrated camera, and modest computational capacity. The ESP32-CAM was selected not only for its affordability and integrated camera but also as a challenging platform that would push the limits of embedded CNN deployment.

- **Analyze and Design the System**: Two different convolutional network (CNN) architectures were selected and explored through separate development modules:
  - **Module 1: MobileNetV2 (Iteration 1):**
    A pre-trained MobileNetv2 was adapted for embedded deployment. Its lightweight architecture and compatibility with quantization made it suitable for real-time inference

    in the ESP32-CAM microcontroller.
  - **Module 2: Custom SimpleCNN (Iteration 2):**
    A custom lightweight CNN architecture (SimpleCNN) was developed using PyTorch. The architecture consists of a minimal number of layers needed to create a working network (2 convolutional layers, each followed by ReLU activation and max pooling), 2 fully connected layers, followed by these. The SimpleCNN was designed as a compromise: it favors low memory usage and speed over model complexity and accuracy. Additionally, it was also designed to suit lightweight embedded inference tasks, such as character recognition on an ESP32-CAM.

- **Build the (Prototype) System**: This phase involves integrating the trained model with the embedded hardware, ensuring compatibility between the software and hardware components, and preparing the system for testing.

- **Observe and Evaluate the System**

The evaluation includes tests of accuracy, response time (inference time), and memory usage on the device. Accuracy was measured programmatically by various handwritten letters to the system and comparing the model's predicted labels with the known ground truth. This automated approach ensured consistent and reproducible evaluation.

Inference time and memory usage were measured using tools provided by Edge Impulse, which offered detailed profiling compatible with the ESP32-CAM hardware.

The results were analyzed relative to the two evaluated models, MobileNetV2 and SimpleCNN, highlighting the trade-offs between model complexity, accuracy, and system performance within the limited memory and processing power available on the ESP32-CAM platform.

4.2 Iteration 1: MobileNetV2 CNN Module

This module investigates the deployment of a pre-trained MobileNetV2 convolutional neural network (CNN) on the ESP32-CAM microcontroller.

- **Dataset:** A reduced version of the EMNIST dataset was used, consisting of approximately 4000 grayscale images representing uppercase letters (26 classes). Each class contained 165 samples. All images were resized to 28x28 using the "fit to shortest" mode to maintain aspect ratios and reduce distortion. The dataset was balanced, with an equal number of images (165) allocated to each uppercase letter (A–Z), minimizing the risk of class imbalance or bias toward specific characters.
- **Architecture:** MobileNetV2 was selected due to its lightweight and modular design, which is particularly suited for mobile and embedded platforms.
- **Preprocessing:** Captured camera frames were converted to grayscale and resized to 28x28 on the ESP32-CAM. These processed images were passed to the CNN model for inference. Preprocessing was implemented directly on the microcontroller.
- **Deployment:** The quantized model was integrated into a firmware project using Arduino IDE. The complete system was compiled and uploaded to the ESP32-CAM, where it was configured to perform inference immediately after each image capture event triggered by the built-in camera.
- **Evaluation:** The system was evaöuated based on its classification accuracy, response time, and memory efficiency when deployed on the ESP32-CAM. Accuracy was determined by presenting various handwritten letters to the device and verifying the model's predictions.

4.3 Iteration 2: Custom Simple CNN Module

This module involves the design, training, and testing of a custom Lightweight CNN (SimpleCNN) model tailored for efficient character classification.

- **Dataset:** The EMNIST Letters dataset [32] consisted of grayscale images of uppercase alphabetic characters (26 classes). Each class label originally ranged from 1 ('A') to 26 ('Z') and was adjusted to follow zero-based indexing to match the model output classes. The dataset was split into training and test subsets for supervised learning and evaluation.

- **Architecture:** The SimpleCNN architecture included two convolutional layers, each followed by ReLU activation and max pooling. Two fully connected layers followed these to perform classification over the 26 letter classes.

This minimalistic design was chosen to ensure compatibility with the ESP32-CAM's resource constraints.

- **Training:** The model was trained using the Adam optimization algorithm with a learning rate of 0.001 and a batch size of 64. Cross-Entropy Loss was used as the loss function. Training was performed over 10 epochs using the training portion of the EMNIST Letters dataset.
- **Preprocessing:** Each input image was resized to 28×28 pixels and converted to grayscale, ensuring consistency with the input shape expected by the model. Label values were adjusted by subtracting 1 from the original EMNIST labels to match the model's zero-based output indexing. This preprocessing was consistently applied during both training and inference.
- **Deployment:** The trained model was quantized to reduce memory usage and integrated into the ESP32-CAM firmware using the Arduino IDE. The system was configured to perform real-time inference using captured frames from the built-in camera, following the same preprocessing pipeline as in training.
- **Evaluation:** Model accuracy was assessed using the test subset of the EMNIST dataset. Performance was measured as the proportion of correctly predicted labels out of the total number of test samples, providing an overall metric of classification accuracy.

5. Results

This chapter presents the results and analysis from the development and evaluation of the two iterative modules described in Chapter 4. The purpose of both modules was to implement and assess machine learning models for handwritten letter recognition in an embedded system context. The models were evaluated in terms of classification accuracy, inference time, and compatibility with resource-constrained hardware such as the ESP32-CAM microcontroller.

Module 1: MobileNetV2 (Iteration 1) focuses on the deployment of a quantized MobileNetV2 model on the ESP32-CAM. The results include the embedded systems' performance in real-time inference, memory usage, and prediction accuracy after deployment.

Module 2: Custom SimpleCNN (Iteration 2) involves the design and training of a custom SimpleCNN model, evaluated first in a simulated environment using a Python-based GUI. Results include classification performance, training behaviour, and usability observations. Finally, the model was evaluated after being deployed on the ESP32-CAM to assess its real-world performance.

The comparative evaluation of both approaches provides insights into the trade-offs between model complexity, real-time feasibility, and deployment constraints, contributing to design decisions for embedded ML applications.

5.1 Iteration 0

5.1.1 Hardware Consistency Across Iterations

Both iteration 1 (MobileNetV2) and iteration 2 (SimpleCNN) use the same hardware configuration. The entire experimental setup, including the ESP32-CAM microcontroller, FTDI programmer for communication, and an OLED display for output, was kept constant to ensure a fair comparison between the different model implementations.

5.1.2 Architecture

The initial phase of this project involved conducting a literature review to determine the technical feasibility of deploying machine learning models on microcontrollers, particularly the ESP32-CAM. The review highlighted several limitations commonly associated with such hardware, including constrained memory, low processing power, and limited storage capacity [28].

In selecting suitable hardware for deployment, the ESP32-CAM microcontroller was chosen due to its widespread availability, low cost, integrated Wi-Fi and Bluetooth modules, and community support. The ESP32-CAM is constrained by its limited RAM of only 520KB, a dual-core CPU,

and restricted flash storage of 4MB [29]. These hardware limitations present major obstacles for deploying machine learning models [30].

The complete hardware setup, shown in Figure 4, includes the ESP32-CAM, an FTDI programmer, and an OLED display.

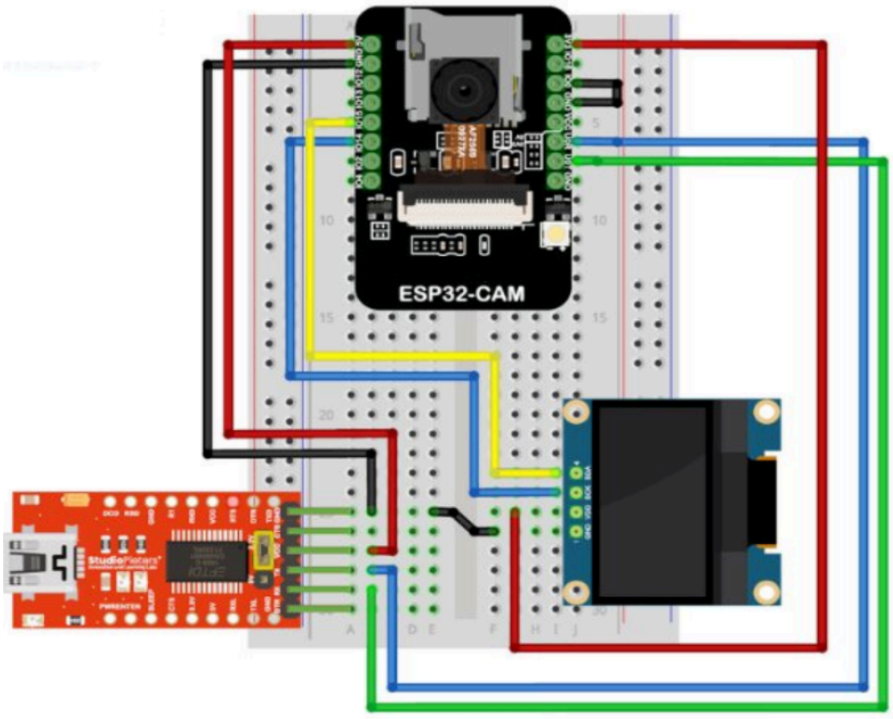

Figure 4: Complete Electrical Schematic

Figure 4 illustrates the Complete Electrical Schematic on paper before assembling. To interface with the ESP32-CAM (center component) during development and uploading of firmware, an FTDI programmer (red component on the left) was used to provide serial communication and power, since this version of the ESP32-CAM lacks a native USB interface.

An OLED display was included in the hardware setup to provide user feedback and display the recognition output in real time. This display shows the predicted letter detected by the system.

During training, all preprocessing, such as grayscale conversion, normalization, resizing, and data augmentation, was carried out offline in the training environment. In contrast, during inference on the ESP32-CAM, the system performs real-time image capture, resizes the input to the required dimensions, and runs the prediction using the trained model.

5.1.3 Design

The circuit design was focused on ensuring reliable communication and minimal latency:

- The camera module captures 28x28 grayscale images.
- The OLED screen is connected to the ESP32-CAM to display classification results.

- The FTDI programmer is connected to the ESP32-CAM UART pins to upload firmware and power the circuit.

Pin selection was based on ESP32-CAM datasheets to avoid conflicts with internal peripherals.

5.1.4 Implementation

The implementation involved:

- Flashing firmware that includes the quantized CNN model
- Initializing camera, I2C (for OLED), and UART serial communication.
- Embedding the trained model into the ESP32-CAM project using ARDUINO IDE or PlatformIO.

For both iterations, the ESP32-CAM runs entirely standalone, performing all operations locally without external computation.

All preprocessing steps, including normalization and augmentation, were performed offline during the training phase. On the ESP32-CAM, the only on-device preprocessing includes real-time grayscale image capture and fixed-size resizing before feeding into the model

5.1.5 Test

Each model was tested on a set of handwritten letters representing all 26 uppercase classes, with the same test samples used across both iterations to ensure a fair comparison. Testing was conducted by placing handwritten letters in front of the ESP32-CAM. The camera captured each image, which was preprocessed and passed to the model for classification.
To support efficient inference, images captured by the camera are resized and preprocessed to match the input requirements of the CNN. This includes grayscale conversion, normalization, and dimensional reduction. These steps are necessary to reduce memory load while maintaining classification accuracy. Lastly, the OLED displayed the predicted letter. Tests were performed under various lighting and background conditions to evaluate robustness. Test samples are shown in Figure 5 below, and Figures 6 and 7 demonstrate the testing results from both iterations.

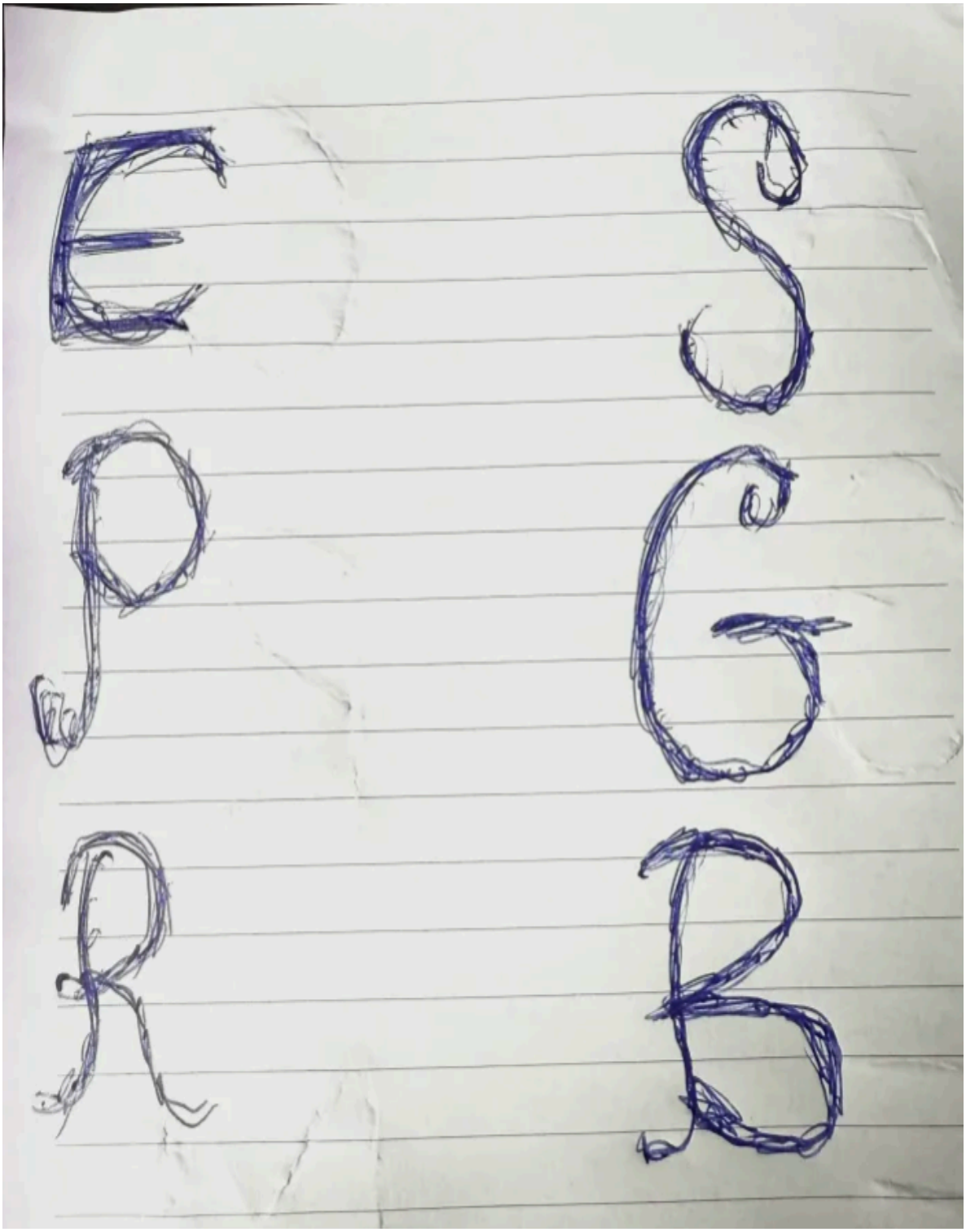

Figure 5: Testing samples

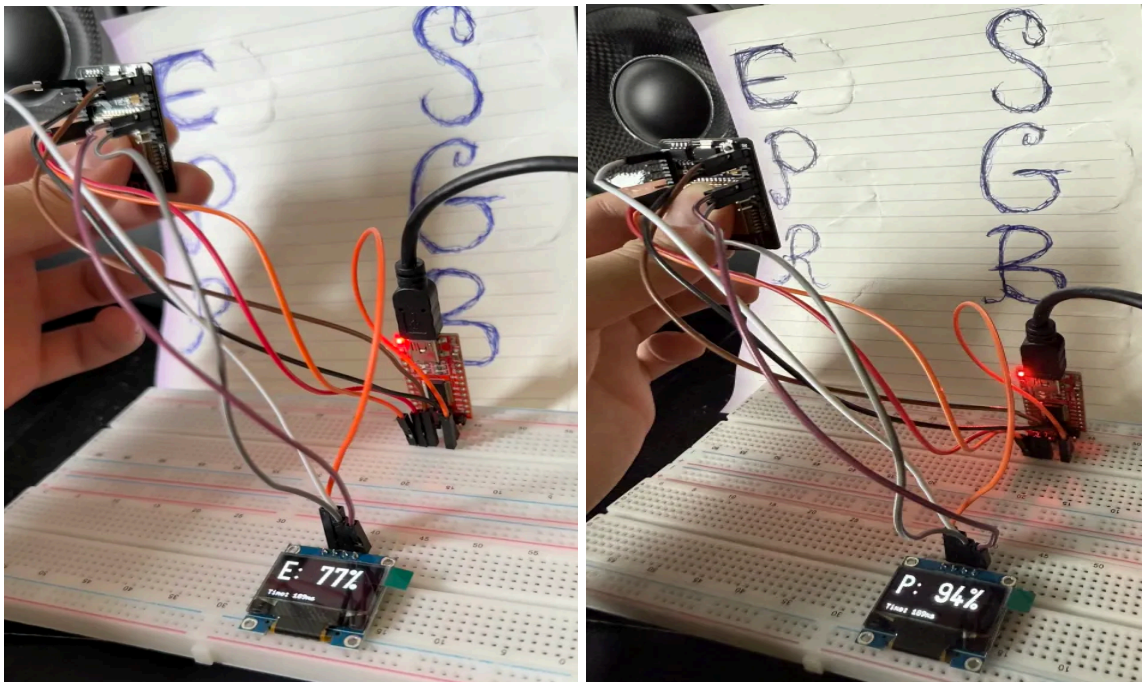

Figure 6: Testing results from Iteration 1 (MobileNetV2)

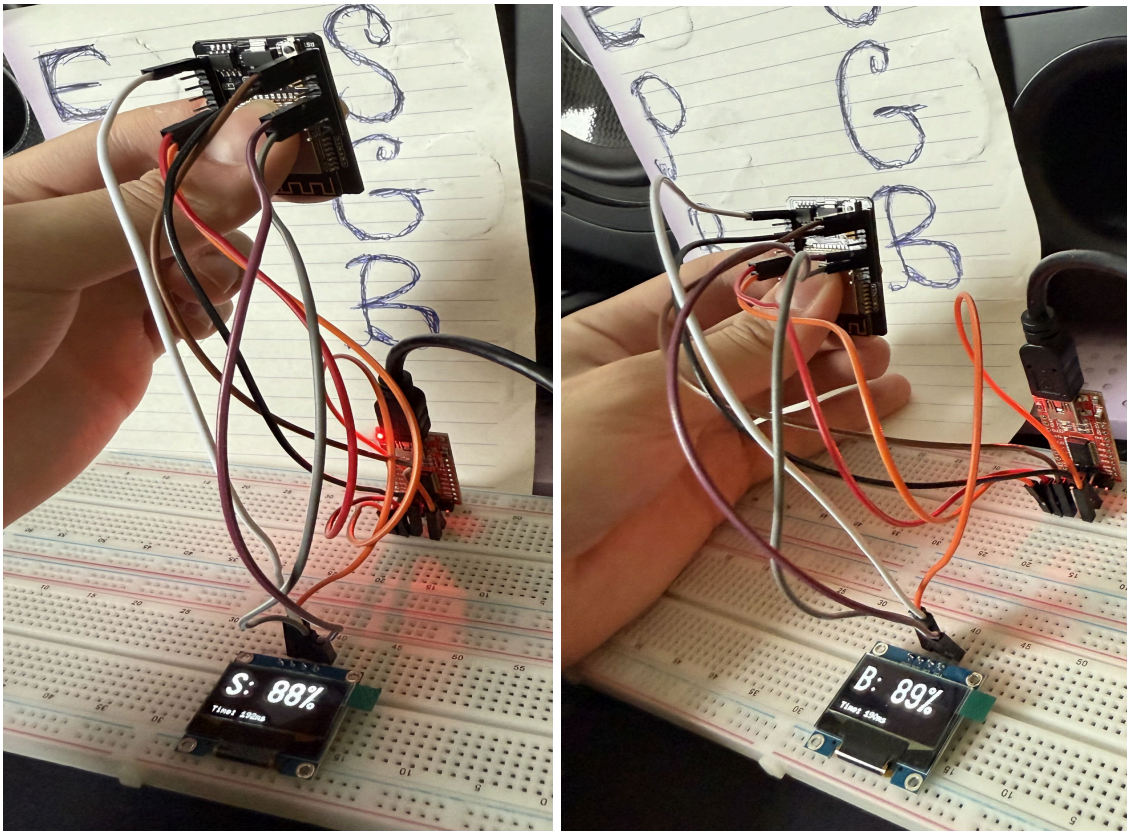

Figure 7: Testing results from Iteration 2 (SimpleCNN)

5.1.6 Evaluation

The evaluation showed the system was functionally complete:

- The ESP32-CAM consistently initialized and captured images correctly.
- The OLED screen displayed outputs with minimal delay
- Misclassifications occurred primarily in visually similar characters.
- The system demonstrated successful end-to-end functionality and reliability across both model deployments.

5.2 Iteration 1

5.2.1 Design of MobileNetV2 architecture

Given the limitations of the ESP32-CAM, it was essential to select a model architecture that balances performance and computational efficiency. Based on related work, Convolutional Neural Networks (CNNs) were identified early in the project as a suitable class of models for image classification tasks, such as handwritten character recognition. CNNs are widely regarded for their ability to extract spatial hierarchies of features from images and have shown superior performance in computer vision tasks [14].

However, not all CNN architectures are compatible with microcontroller-level hardware. Lighter-weight models of CNN, such as MobileNetV2 [31], were considered and ultimately chosen as it was proven to work on low-memory embedded systems. The model offers a good tradeoff between inference accuracy and model size [31].

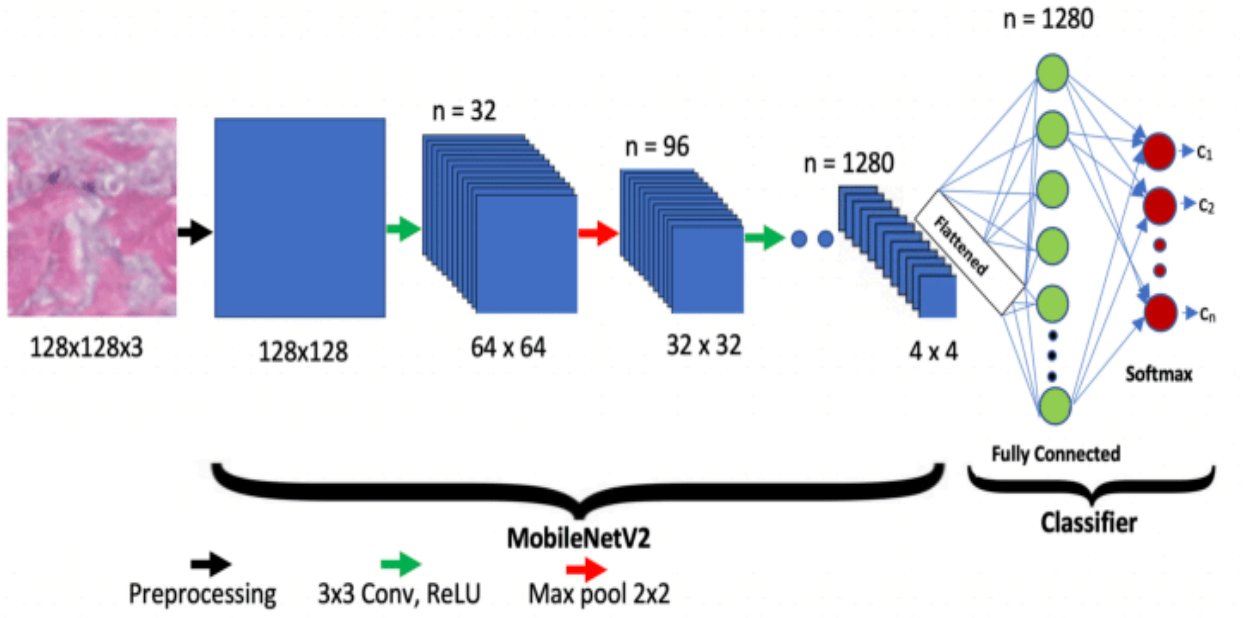

Figure 8: MobileNetV2 network architecture

MobileNetV2 is a lightweight CNN optimized for mobile and embedded vision applications. It is composed of 53 layers grouped into 17 blocks, employing depthwise separable convolutions to significantly reduce computation without sacrificing performance. As shown in Figure 8, each block consists of a sequence of layers known as inverted residuals. Each block consists of a sequence of layers known as inverted residuals with linear bottlenecks. These bottleneck blocks include a 1x1 convolution (expansion layer), a depthwise 3x3 convolution, and another 1x1 convolution (projection layer). The ReLU activation function is used in the intermediate layers, except for the final projection, where a linear activation is applied to preserve low-dimensional features [32].

As described in the source document [32], the network utilizes a contracting path for feature extraction, which is followed by a classifier head. The contracting path involves repeating blocks of convolution, activation, and max-pooling operations, doubling the number of feature channels at each downsampling stage. Finally, a fully connected layer with a softmax activation function is used for classification. This design enables efficient training and inference on devices with limited resources.

5.2.2 Dataset preparation

A reduced version of the EMNIST dataset [33] was used to support the development and training of the system. The original dataset contained samples of both uppercase and lowercase letters, as well as digits. However, to meet the memory and computational constraints of the ESP32-CAM, the dataset was narrowed down to include only uppercase letters (A-Z), with approximately 165 images per class, totaling around 4000 grayscale images. This reduction ensured that the resulting model remained lightweight and compatible with the target hardware. The trained model in quantized int8 format was approximately 10 MB, making it suitable for deployment on the ESP32-CAM.

5.2.3 System Design and Analysis

This phase focused on refining the system design to meet the specific constraints posed by the microcontroller and improving the model's performance through iterative evaluation, which led to a final F1-score of 89% and reduced inference latency to under 300 ms on the ESP32-CAM. Throughout the development, several training configurations were tested to assess the impact of different parameters such as learning rate, number of epochs, and whether data augmentation was applied.

5.2.3.1 Model Training and Analysis

The benchmark criteria included F1 score, number of training epochs, and inference suitability for the ESP32-CAM. Initially, training was attempted on a larger dataset, approximately 10,000 images with both letters and digits, uppercase and lowercase. Due to resource limitations from Edge Impulse only allowing training lower than 20 minutes, only training with up to 8 epochs was possible at most. This resulted in an F1 score of only 48.4%.

F1 SCORE
48.4%

	F1-SCORE	PRECISION	RECALL
BACKGROUND	1.00	1.00	1.00
0	0.11	0.12	0.10
1	0.09	1.00	0.05
2	0.35	0.88	0.22
3	0.56	1.00	0.39
4	0.06	1.00	0.03
5	0.67	1.00	0.50
6	0.73	0.82	0.67
7	0.37	0.46	0.32
8	0.06	1.00	0.03
9	0.00	0.00	0.00
A	0.26	0.50	0.18
B	0.46	0.44	0.48
C	0.67	0.64	0.70
D	0.34	1.00	0.21
E	0.57	0.90	0.42
F	0.57	0.59	0.55
G	0.19	1.00	0.11
I	0.34	0.29	0.42
J	0.30	0.37	0.25
K	0.68	0.93	0.53
L	0.00	0.00	0.00
M	0.81	0.79	0.83
N	0.29	0.75	0.18
O	0.38	0.67	0.27
P	0.27	0.73	0.16
Q	0.29	0.50	0.20
R	0.11	1.00	0.06
S	0.56	0.69	0.47
T	0.35	0.88	0.22
U	0.74	0.73	0.75
V	0.76	0.80	0.73
W	0.73	0.76	0.70
X	0.57	0.90	0.41
Y	0.70	0.82	0.62
Z	0.42	1.00	0.26
H	0.76	0.83	0.69

Figure 9: Training result table on the EMNIST dataset

Looking at Figure 9, the training result table reveals that the model struggled significantly across most character classes. Many characters exhibit very low F1 scores, highlighting difficulties in distinguishing between visually similar characters.

Classes such as L performed poorly, likely due to visual similarity to other classes. Letters M, U, and H achieved strong F1 scores, indicating more distinctive features or better data representation. Precision was occasionally high, but often paired with very low recall, suggesting that the model correctly predicted a few instances while failing to generalize.

The low number of training epochs appears insufficient for the model to fully learn the complex patterns required to distinguish a wide range of handwritten letters.

Figure 10: Inference performance metrics (EMNIST dataset)

As demonstrated in Figure 10, the model was evaluated on the ESP32-CAM using the EON Compiler provided by the Edge Impulse platform. The system demonstrated a peak RAM usage of approximately 120 KB, with flash memory consumption of 92.6 KB, and the average inference time was 1327 milliseconds.

Due to the unsatisfactory results from the initial training phase using the full dataset, the dataset was intentionally reduced to include only uppercase letters (A–Z), with 165 images per class. This reduction enabled longer training cycles (up to 25 epochs), improved the model's ability to generalize, and ensured that the final model was small enough for deployment on the ESP32-CAM.

Multiple training configurations were tested, varying key hyperparameters such as the number of training epochs, learning rate, and whether data augmentation was applied (see Table 1 for details). These variables were selected because they directly influence model accuracy, generalization, and training efficiency—factors critical for achieving good performance under the memory and computational constraints of the ESP32-CAM.

Table 1 (Training configurations and results with reduced dataset):

Number of Epochs	Learning Rate	Data Augmentation	F1 Score
10	0.005	ON	59.2%
12	0.005	ON	68.2%
14	0.007	ON	72.0%
15	0.003	ON	68.7%
15	0.010	ON	69.7%
15	0.005	ON	75.2%
15	0.004	ON	75.6%
16	0.005	OFF	71.3%
16	0.005	ON	74.0%
17	0.007	ON	74.7%
20	0.001	ON	53.0%
25	0.005	ON	87.4%

	F1-SCORE	PRECISION	RECALL
BACKGROUND	1.00	1.00	1.00
A	0.90	0.96	0.85
B	0.80	0.83	0.77
C	0.98	0.97	1.00
D	0.81	0.68	1.00
E	0.94	0.89	1.00
F	0.95	1.00	0.91
G	0.64	0.47	1.00
H	0.91	1.00	0.83
I	0.72	0.78	0.67
J	0.89	0.87	0.90
K	0.90	0.90	0.90
L	0.91	0.89	0.92
M	0.92	0.87	0.96
N	0.86	0.78	0.95
O	0.94	0.96	0.92
P	0.94	0.88	1.00
Q	0.92	0.96	0.88
R	0.83	0.73	0.95
S	0.84	0.89	0.80
T	0.95	0.93	0.96
U	0.98	1.00	0.96
V	0.88	0.79	1.00
W	0.86	0.86	0.86
X	0.84	0.74	0.97
Y	0.88	0.88	0.88
Z	0.86	0.83	0.90

Figure 11: Training results of the highest F1 score (reduced dataset)

As shown in Table 1, the best-performing configuration from the testing phase includes 25 epochs, a 0.005 learning rate, and data augmentation enabled. This configuration, demonstrated in Figure 11, yielded a significant improvement in classification performance, achieving an

overall F1 score of 87.4%. This marks a substantial increase from the original dataset's F1 score of only 48.4%, affirming the advantage of a more focused dataset and extended training.

5.2.3.2 On-Device Performance

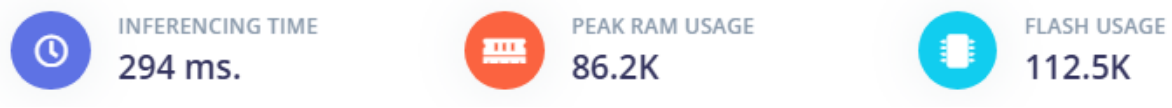

Figure 12: Inference performance metrics (Reduced dataset)

Figure 12 shows that the model trained on the reduced dataset achieved a dramatically lower inference time of 294 milliseconds with peak RAM usage reduced to 86.2KB and flash usage increased slightly to 112.5 KB. This reflects the improved responsiveness and compact memory footprint of the reduced dataset model.

Overall, the reduced dataset allowed the model to concentrate on a narrower but more relevant range of input classes. This not only facilitated deeper learning of class-specific features through increased epochs but also enabled a more responsive system design suitable for real-time applications on embedded hardware, without significant compromise in accuracy.

5.2.4 Build the (Prototype) System

The implementation phase involved constructing a complete prototype system that integrates both hardware and software components to enable real-time handwriting recognition on the microcontroller.

The trained MobileNetV2 model was exported in quantized int8 format and integrated into the ESP32-CAM using the Arduino IDE environment and C++. The firmware was programmed to:

- Initialize the camera on boot.
- Continuously capture images at fixed intervals.
- Preprocess the images, resizing and formatting them to match the model's input requirements.
- Run inference on the captured frame using the deployed model.
- Print prediction results to the OLED screen.

By default, the system captures and classifies a new image every 5 milliseconds, allowing for rapid and responsive prediction cycles. This loop ensures real-time performance, making the prototype responsive to changes in input.

The system is fully self-contained and does not rely on any external devices or connectivity. All computations are performed locally on the microcontroller itself.

5.2.5 Observe and Evaluate the System

5.2.5.1 Training Metrics Analysis

To better understand the learning behavior of the model, the training and validation loss curves were analyzed. It is visually illustrated in the figure below:

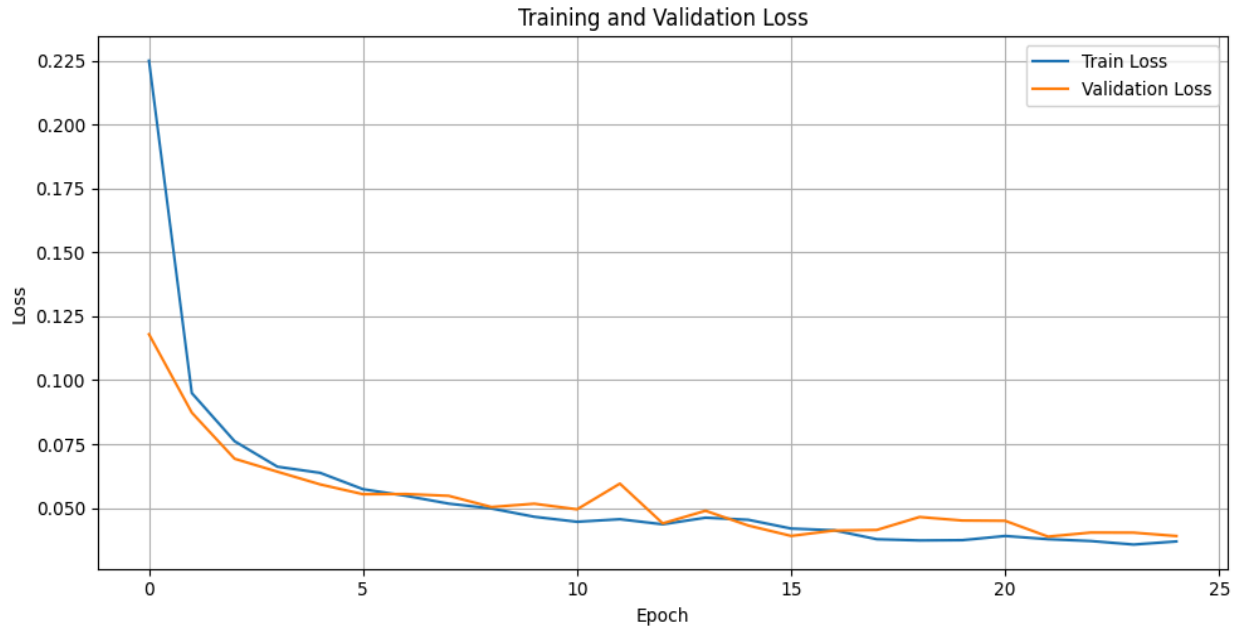

Figure 13: Training and validation loss over 25 epochs

The training loss showed a rapid decline in the first few epochs, indicating successful convergence. Validation loss followed a similar trajectory with minor fluctuations, suggesting that the model generalized well without significant overfitting. The close alignment between the two curves across 25 epochs confirms that the chosen training configuration was effective in balancing fit and generalization.

Further analysis was conducted using per-class recall values extracted from the validation results, as shown in Figure 14 below:

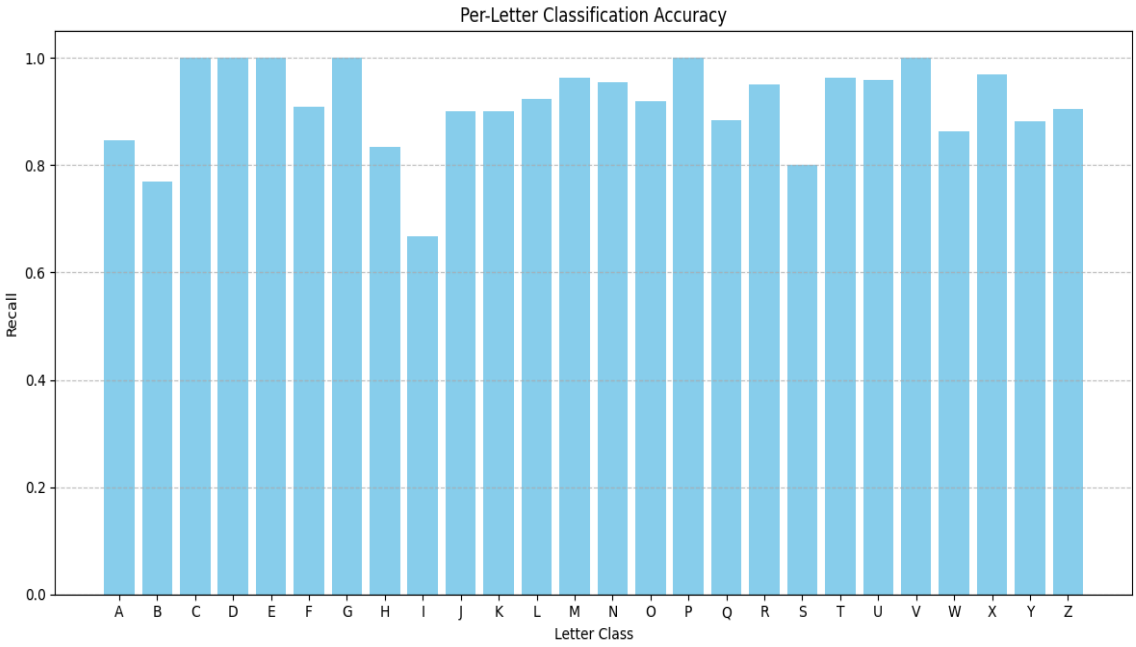

Figure 14: Per-letter classification accuracy

The overall pattern indicates that characters with simpler or more symmetrical shapes tend to be recognized with higher confidence, while those with ambiguous curves or common structural elements suffer from misclassification. This aligns with the expectations for a CNN model like MobileNetV2, which makes trade-offs in representational complexity to fit microcontroller constraints.

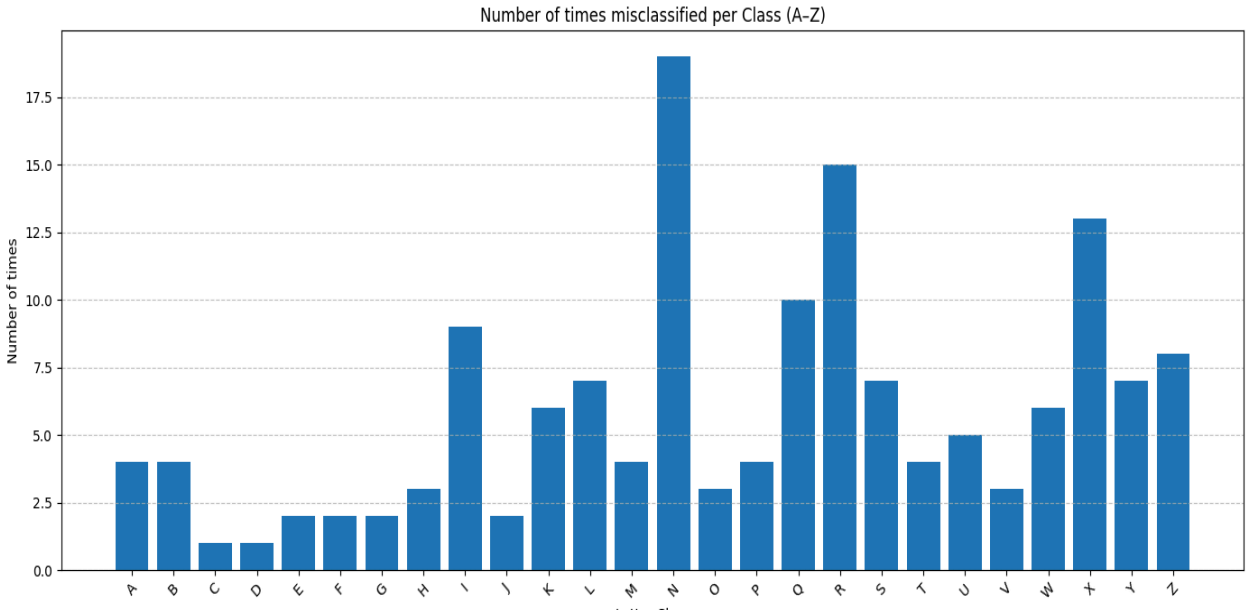

Figure 15: Number of times misclassified for each class

The bar chart in Figure 15 shows how often each letter from A to Z was misclassified by the model, based on the sum of false positives and false negatives. This helps identify which letters the system struggles with most.

The letter N stands out clearly with the highest number of misclassifications. Letters R, X, and Q also show high error rates compared to the rest. These results suggest that these letters may be visually similar to others, making them harder for the model to distinguish. For example, N and M share structural similarities, as do R and P, and Q can often resemble O depending on handwriting.

On the other hand, letters like C, D, and J show very few misclassifications. These likely have more distinct shapes that the model can recognize with greater confidence.

The overall pattern indicates that while the model performs well for many classes, certain letters consistently lead to confusion. Improving accuracy for these letters could involve collecting more diverse training examples or enhancing preprocessing techniques such as edge detection or normalization. Analyzing which letters are being confused with each other would also provide deeper insight and help guide further refinement of the model. This type of analysis is useful for understanding real-world performance and ensuring more balanced recognition across all character classes.

5.2.5.2 System Evaluation

Following the integration of the hardware components and the deployment of the quantized MobileNetV2 model onto the ESP32-CAM, a series of evaluations were conducted to assess the system's functionality in practical settings. This evaluation phase was used to verify whether the system could operate autonomously, maintain inference accuracy, and deliver stable performance under typical real-world conditions. The aims were to confirm the model's prediction reliability, observe system responsiveness during continuous operation, and explore how the embedded setup handles constraints inherent to low-power hardware.

Once the physical assembly was completed and the system was powered on, the ESP32-CAM initialized all necessary peripherals, including the onboard camera module and the display. Upon initialization, the system immediately began executing its primary task loop of continuously capturing grayscale images, preprocessing, passing them through the trained CNN, and displaying the result on the screen in real time.

Tests were conducted by positioning handwritten letters in front of the camera at varying orientations and distances. The inference loop processed each image in approximately 150 milliseconds, demonstrating that the system was capable of near real-time interaction. In each cycle, the classification of prediction correctness is performed. During the test procedure, particular attention was given to the system's ability to sustain performance across multiple inputs, ensuring that it could classify a sequence of characters without requiring reinitialization or manual intervention.

In practice, the system produced correct classifications for a majority of letters with a high degree of consistency. The performance aligned well with the training results and confusion matrices. Certain letters, such as ‘I’,’ G’, ‘R’, ‘D’, and letters that look similar to those four letters, were occasionally misclassified.

Despite the microcontroller’s limited computational capacity, including constraints on RAM and flash storage, the deployed model maintained a high overall F1 score of 87.4%. This score reflects a balanced trade-off between precision and recall, indicating that the system performed reliably across the full range of test letters. Furthermore, this evaluation confirms that the inference process remained stable throughout operation without crashing.

In summary, the test results suggest that it is possible to run a moderately complex machine learning model on the ESP32-CAM for real-time handwritten letter recognition. The system’s behavior under deployment conditions indicates that the trained model performs reliably and that the hardware-software integration functions as intended. These outcomes support the potential of lightweight neural architectures like MobileNetV2 in embedded AI applications, especially in scenarios where resource efficiency and practical accuracy must be balanced.

5.2.6 Confusion Matrix

Most character classes exhibit a high F1 score, precision, and recall. For instance, letters such as T, U, and C achieved nearly perfect recall values of 1.00, suggesting that the model was consistently able to correctly identify instances of these classes. The F1 scores for most letters range between 0.88 and 0.98, indicating a strong balance between precision and recall.

However, there remain some character classes that pose challenges. The letter G had an F1 score of only 0.64 and a precision of 0.47, indicating frequent misclassifications. This may be attributed to the visual similarity of G to other rounded characters, leading to confusion during inference. Similarly, the letter I recorded a comparatively low recall of 0.67, suggesting that the model occasionally failed to detect true instances of I.

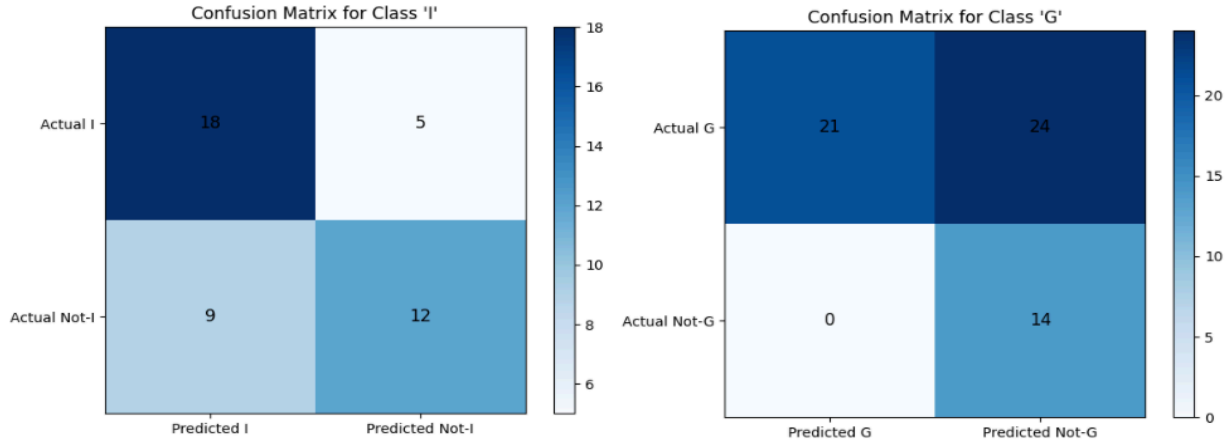

Figure 16: Confusion matrix for letter I and G

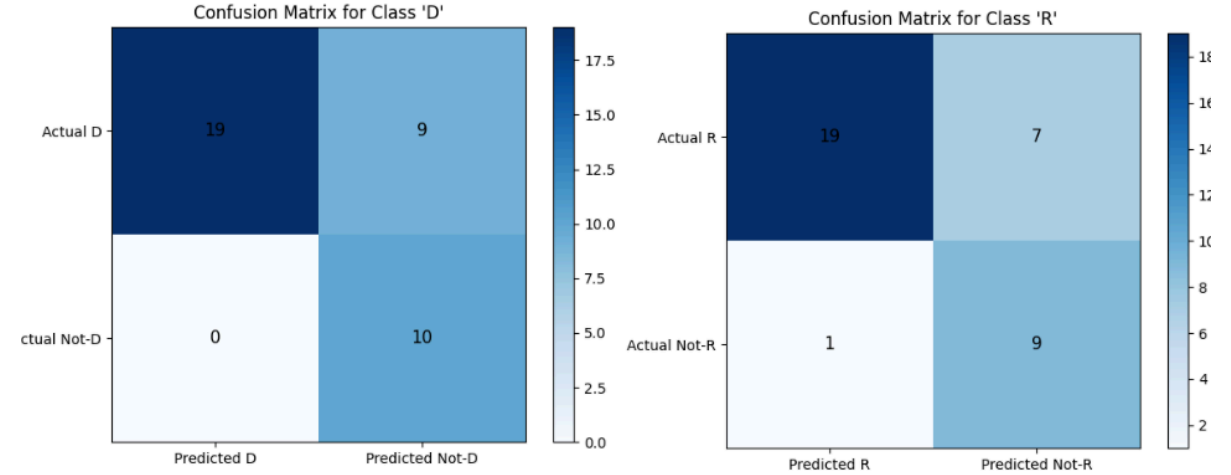

Figure 17: Confusion matrix for letter D and R

Figures 16 and 17 above present the detailed confusion matrices for the 4 lowest-performing letters, D, G, I, and R, offering insight into their individual classification performance. Each confusion matrix consists of 4 components: TP are the correctly predicted instances of the class, FP are incorrect predictions where another class was mistakenly labeled as the target, FN represent true instances of the target class that were incorrectly predicted as another class and TN are all other correct predictions not involving the target class.

As an example, consider the confusion matrix for letter G. According to the training logs, the model predicted 21 instances correctly (True Positives TP), but also incorrectly predicted 24 other letters as G (False Positives FP). There were no False Negatives in this case, indicating the model did not miss any actual G instances. Applying the formulas from section 2.10:

- **Precision** $= \frac{TP}{TP + FP} = \frac{21}{21 + 24} = \mathbf{0.467}$ **(1)**
- **Recall** $= \frac{TP}{TP + FN} = \frac{21}{21 + 0} = \mathbf{1.00}$ **(2)**
- **F1 Score** $= 2 \bullet \frac{Precision \bullet Recall}{Precision + Recall} = 2 \bullet \frac{0.467 \bullet 1.00}{0.467 + 1.00} = \mathbf{0.64 = 64\%}$ **(3)**

Despite achieving perfect recall, the low precision significantly reduced the overall F1 score for the letter G. This means the model successfully detected all instances of the letter G but was overly eager in labeling other characters as G, resulting in many false positives. Consequently, this led to a low F1 score of 0.64.

A similar pattern can be observed in the other letters with low performance. For instance, letter D had 19 TP but also 9 FP, while letter I suffered from both a relatively high number of FN (9) and FP (5), reducing both its precision and recall. The letter R also had a high number of incorrect classifications, with 7 FP and 1 FN.

In all these cases, the drop in F1 score stems from a combination of high false positives, false negatives, or both. This illustrates a common limitation in multiclass classification tasks, especially when deploying on constrained devices like the ESP32-CAM: the model tends to confuse characters that are visually similar or underrepresented, which undermines class-specific reliability even when the overall accuracy remains high.

5.3 Iteration 2

The section presents the development and evaluation process of the second iterative module, which explores a custom-designed SimpleCNN architecture for handwritten letter recognition. The structure follows the same methodology as applied in iterative method 1, including literature review, system design, implementation, and evaluation phases. However, the selection and design of the model architecture differ significantly, as the SimpleCNN was created as the SimpleCNN architecture was independently designed using standard deep learning libraries, with a focus on minimizing the size of the trained model.

5.3.1 Design of Custom SimpleCNN Architecture

The CNN architecture was implemented in a Python script named “network.py”. While the file name is arbitrary, it is referenced consistently throughout this project for clarity.. It consisted of:

- Input: 1 Channel (grayscale) image.
- Conv Layer 1: nn.Conv2d(1, 32, kernel_size = 3, padding = 1) followed by ReLU and 2×2 MaxPooling
- Conv Layer 2: nn.Conv2d(32, 64, kernel_size = 3, padding = 1) followed by ReLU and 2×2 MaxPooling
- Fully connected Layer 1: Linear ($64 \times 7 \times 7$, 128) + ReLU
- Fully Connected Layer 2: Linear (128, 26) that outputs 26 classes (A-Z)

The architecture is named SimpleCNN in the implementation because it is a simple custom-written model, not a pre-trained model like MobileNetV2 and other types. A visualization diagram for the architecture is shown in Figure 18 below.

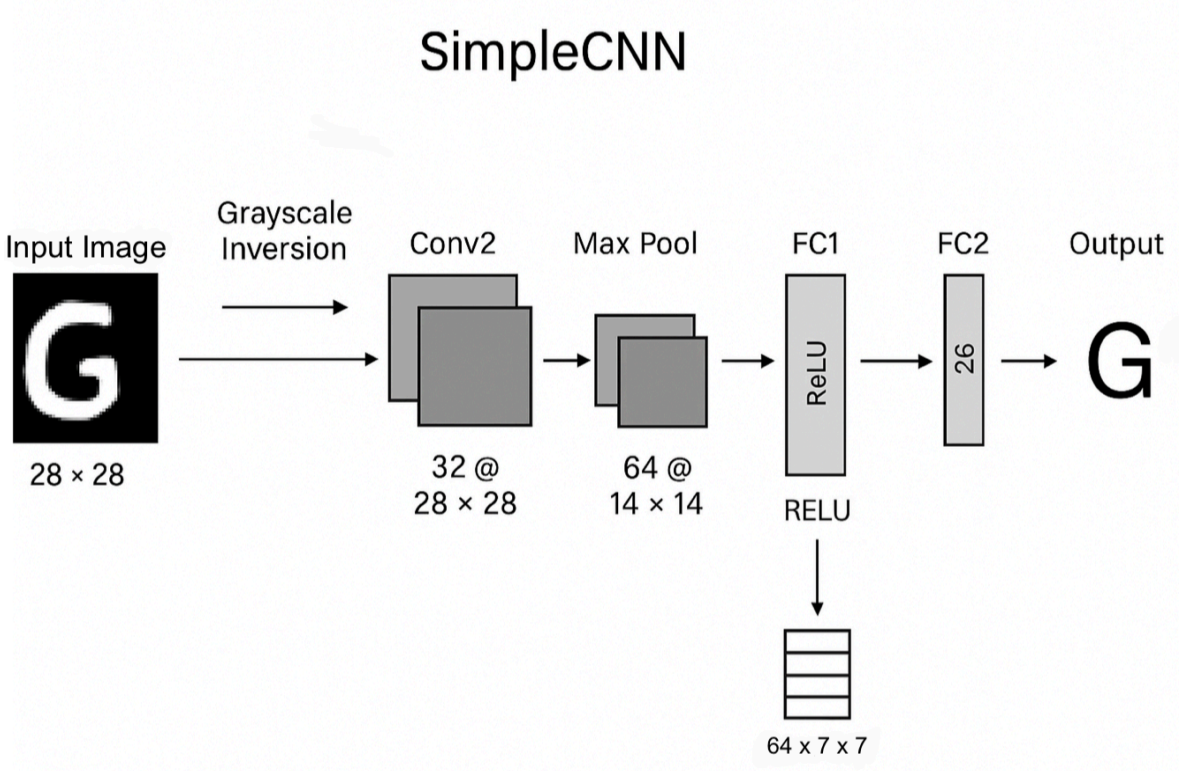

Figure 18: SimpleCNN Architecture

5.3.2 Dataset Preparation

The EMNIST letters dataset [32] was downloaded via the "torchvision.datasets" API with the letters split. Images were transformed using ToTensor() and normalized to the range [-1,1] using a mean and standard deviation of 0.5. The dataset was divided into training and test sets according to the original split. Each of the 26 letters is represented by approximately 4,800 samples, ensuring a balanced distribution across classes

5.3.3 System Design and Analysis

The design of the system revolved around creating a lightweight CNN architecture. The goal was to develop a network that could be potentially portable to embedded systems in future iterations. The SimpleCNN model was constructed with two convolutional layers (each followed by ReLU activation and max pooling) and two fully connected layers for classification into 26 letter categories (A-Z). Preprocessing included grayscale conversion, normalization, and resizing of the input image to 28×28 pixels to match the EMNIST format. The idea behind the model was to minimize the number of layers and parameters while preserving classification accuracy. The architecture was iteratively adjusted and tested through hyperparameter tuning, including learning rate, batch size, and number of epochs.

5.3.3.1 Detailed Code Explanation

Figure 19 defines a convolutional neural network (CNN) in PyTorch, designed for the classification of handwritten English capital letters (A-Z). The implementation begins by importing essential libraries: "torch" and "torch.nn" for defining and training the model, "torchvision.transforms" for image preprocessing, and "PIL.ImageOps" to apply grayscale conversion and color inversion on input images.
The core component is the SimpleCSS class, which inherits from "torch.nn.Module", making it a valid PyTorch neural network model. The network architecture consists of two convolutional layers followed by two fully connected layers. The first layer accepts a single-channel grayscale image and applies 32 convolutional filters of size 3×3, with padding of 1 to preserve the spatial dimensions of the input 28×28. This layer is followed by a max pooling layer with a 2×2 window, which reduces the spatial size by half (28×28 to 14×14). The second convolutional layer applies 64 filters of size 3x3 again with padding, followed by another max pooling layer to further reduce the spatial dimension $\rightarrow$ (7×7). As a result, the final feature map has a $64 \times 7 \times 7$ dimension.

The extracted features are then flattened and passed through a fully connected layer with 128 hidden units. A final linear layer projects this to a 26-dimensional output, corresponding to the 26 letters of the English alphabet. Rectified Linear Unit (ReLU) activation functions are used after both the convolutional and first fully connected layers to introduce non-linearity into the model. The final output layer provides raw prediction scores (logits) for each class.

Once the model architecture is defined, the code checks for hardware availability and moves the model to the GPU if available; otherwise, it defaults to CPU. The pre-trained model weights are loaded from a file named "model_v1.pth" using "load_state_dict()". The model is then set to evaluation mode using "model.eval()" to disable layers like dropout or batch normalization that behave differently during training and inference.

To prepare input images for classification, a transformation pipeline is defined using "torchvision.transforms.Compose". This pipeline resizes the image to 28×28 pixels, then converts it to a tensor, and lastly normalizes the pixel values to fall within the range [-1, 1], using a mean of "0.5" and a standard deviation of "0.5". These steps ensure compatibility with the model's training data format. In addition, they improve generalization during inference.

The "predict_image()" function is responsible for handling the input image and performing the prediction. It accepts a PIL image as input, converts it to grayscale (ensuring a single channel), and inverts the pixel values. This inversion step is crucial if the model was trained on data with dark foreground on a light background, which is common in handwritten digit and character datasets. The preprocessed image is then passed through the transformation pipeline and reshaped to include a batch dimension, resulting in a tensor of shape "[1, 1, 28, 28]".

The model processes this tensor and outputs a vector of 26 logits, one for each class. The index of the highest logit value is determined using "torch.max()", and this index is mapped to a corresponding ASVII character by adding the index to the ordinal value of 'A'. The final result is a string representing the predicted letter, which is returned by the function.

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 26)  # 26 classes for A-Z

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        #  conv1(x)
        x = self.pool(torch.relu(self.conv2(x)))
        #  conv2(x)
        x = x.view(-1, 64 * 7 * 7)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Figure 19: The code of the SimpleCNN model

5.3.4 Build the (Prototype) System

The prototype system was implemented in Python using the PyTorch framework on the Visual Studio Code IDE. The model was trained on the EMNIST letters dataset [32] using an 80/20 train-test split. The prototype initially included a graphical user interface (GUI) that allowed users to draw letters or upload test images for real-time predictions. The GUI was developed using the tkinter library, with back-end integration of the trained SimpleCNN model. The prototype allowed rapid evaluation and debugging of model performance without the constraints of embedded memory or computation limits. The system was designed for modularity to allow conversion to TensorFlow Lite.

The prototype was tested on the embedded system (ESP32-CAM). The trained model was converted from PyTorch to ONNX (‘.pth’ → ‘.ONNX’). After that, from ONNX to TensorFlow. Then from TensorFLow to TensorFlow Lite (TFLite). All conversion steps between the types of files are done using Python code. After that, the TFLite model is converted into a C array (.h file). Finally, deploying it into a PlatformIO/Arduino IDE project with TensorFlow Lite for Microcontrollers (TFLM).

5.3.5 Observe and Evaluate the system

Following the completion of the hardware integration and deployment of the trained model onto the ESP32-CAM, the full system was tested under various conditions to assess its practical functionality. The system initializes the onboard camera, captures handwritten input, preprocesses the images, performs inference using the deployed SimpleCNN model, and displays the predicted character on the LCD.

Figure 20 below shows the training loss over 25 Epochs. The training loss steadily decreased, indicating successful learning and convergence. The curve almost flattens after epoch 20, suggesting the model is nearing optimal performance under the current setup. Additionally, Table 2 provides detailed information about the loss per epoch and the final accuracy that the model reached.
During testing, the system demonstrated a high rate of correct predictions across most letters. However, certain characters-particularly 'J', 'I', 'L', and 'Q' were occasionally misclassified as shown in Figure 21.

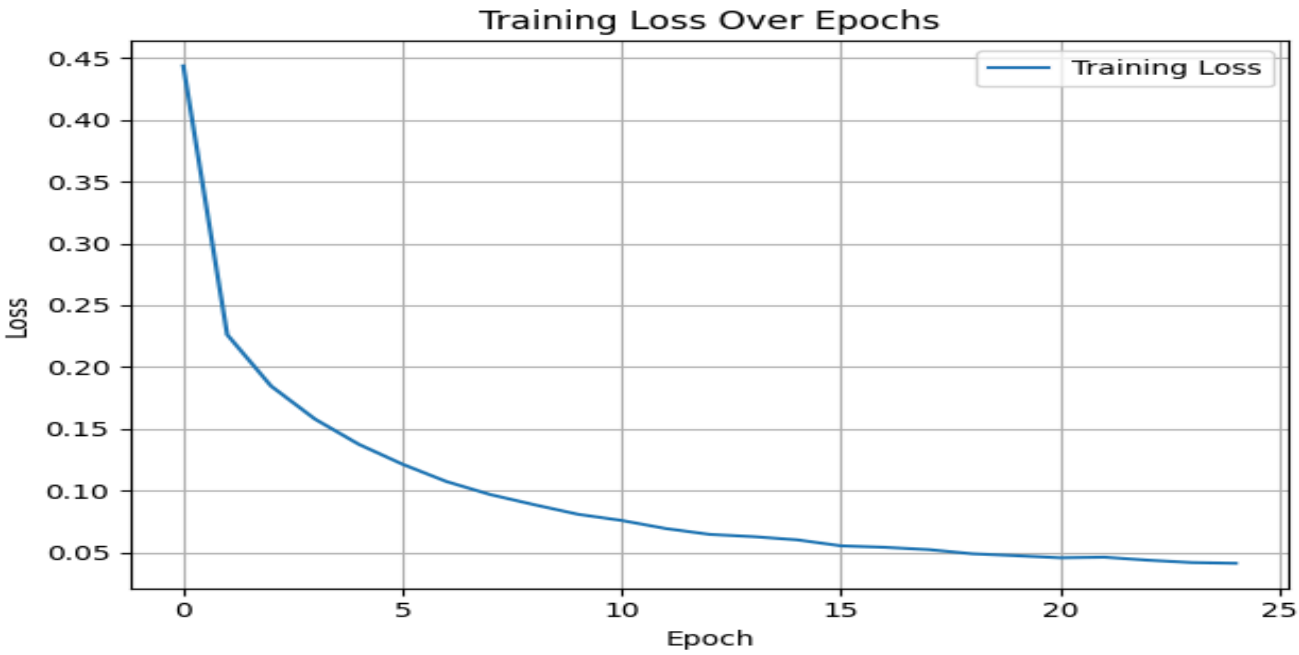

Figure 20: Training loss curve over 25 epochs.

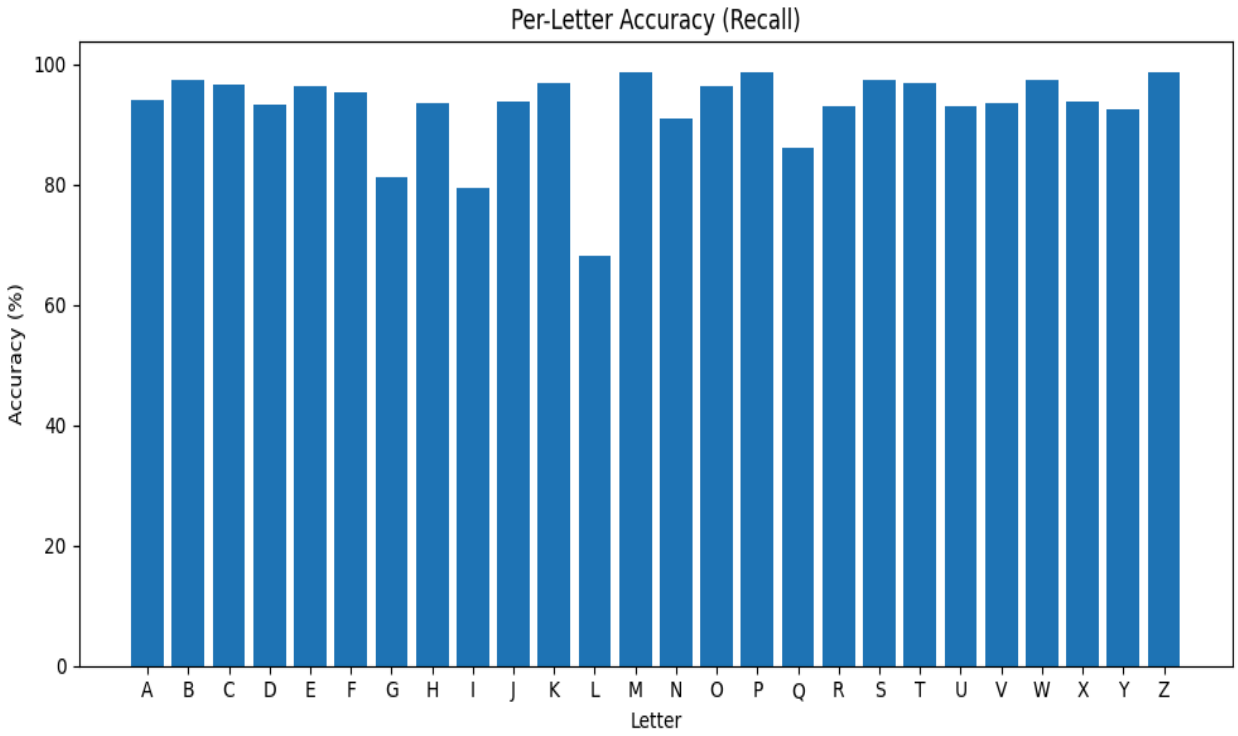

Figure 21: Per-letter classification accuracy. Noticeably low accuracy for letters L, I, G, and Q.

Table 2: Training progress over 25 epochs showing the loss values for each epoch and the model's final accuracy (**92.80%)** that is reached after the 25 epochs

Epoch	**Loss**	**Accuracy**
1	0.4257	92.80%
2	0.2180	
3	0.1781	
4	0.1530	
5	0.1340	
6	0.1171	
7	0.1045	
8	0.0946	
9	0.0853	
10	0.0785	
11	0.0735	
12	0.0669	
13	0.0652	
14	0.0586	
15	0.0573	
16	0.0557	
17	0.0529	
18	0.0513	
19	0.0494	
20	0.0460	
21	0.0448	
22	0.0438	
23	0.0437	
24	0.0427	
25	0.0405	

5.3.6 Confusion Matrix

To further analyze and evaluate the letter-wise classification performance, confusion matrices were generated for a selection of visually similar commonly misclassified letters: ‘L’, ‘I’, ‘G’, and ‘Q’. These matrices are shown in figures 22-25 and provide deeper insights into classification errors, particularly false positives (FP) and false negatives (FN), which are not captured by overall accuracy alone.

Each confusion matrix offers a breakdown of (TP), (TN), (FP), and (FN) for the respective letter, allowing targeted evaluation of the model's ability to distinguish between similar handwritten characters.

Table 3 summarizes the confusion matrix values. The table shows that while overall classification accuracy remains high, precision and recall vary across letters. For example, the letter ‘L’ has lower recall (66.4%) compared to others, indicating a higher rate of missed detections (FN), which is due to its similarity with the uppercase ‘I’ in certain handwriting styles.

Table 3: Confusion matrix Values and Performance for letters L, I, G, and Q.

	Predicted Not	**Predicted**	**Accuracy**	**Precision**	**Recall**
True Not L	19840	160	97.3%	76.8%	66.4%
True L	269	531			
True Not I	19723	277	97.8%	69.7%	79.8%
True I	162	638			
True Not G	19853	147	98.5%	81.9%	83.0%
True G	136	664			
True Not Q	19869	131	98.7%	83.7%	84.4%
True Q	125	675			

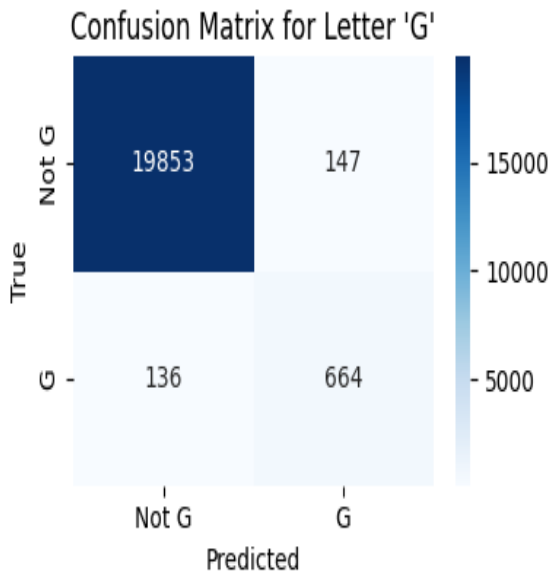

Figure 22: Confusion Matrix for Letter G

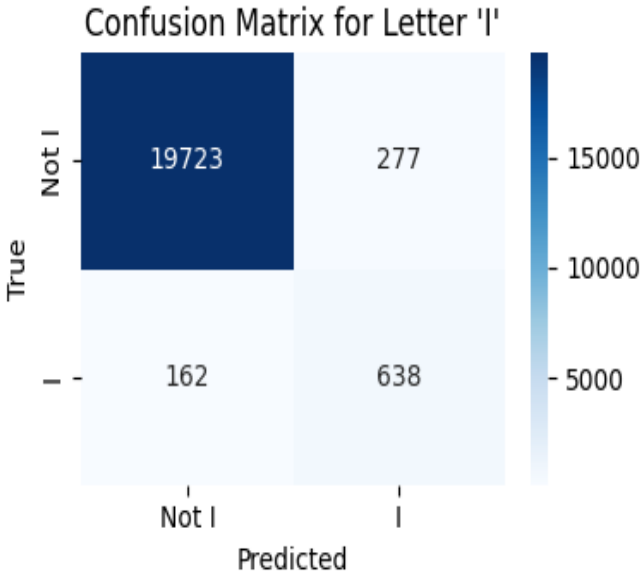

Figure 23: Confusion Matrix for Letter I

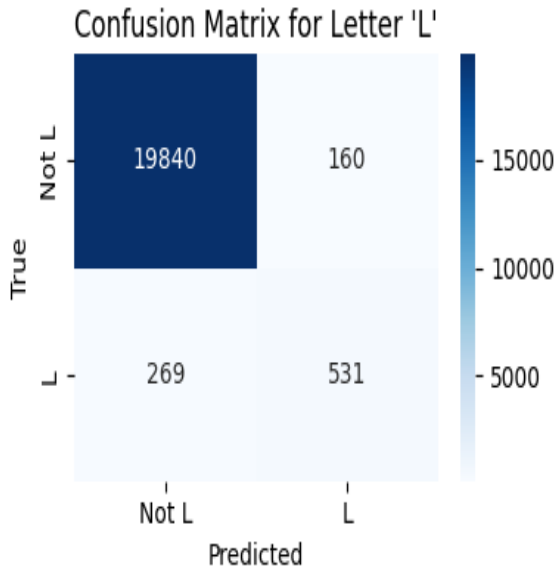

Figure 24: Confusion Matrix for Letter L

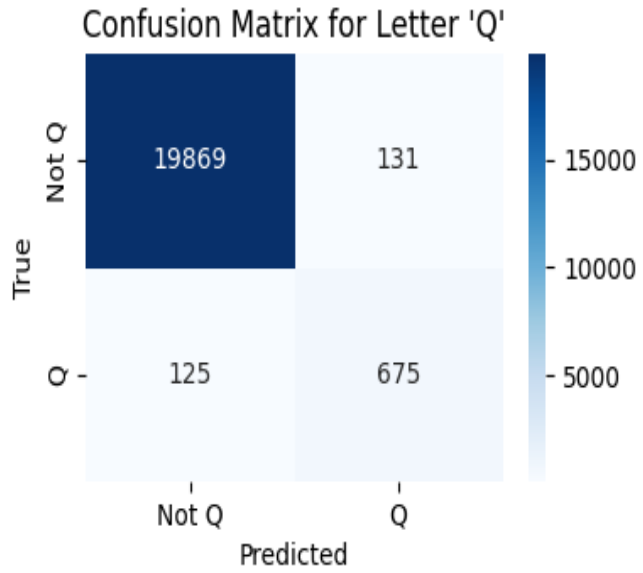

Figure 25: Confusion Matrix for Letter Q

5.3.7 On-Device Performance

On-device performance metrics, shown in Figure 26 below, demonstrate execution suitable for low-power embedded environments. The inference time was approximately 194 ms, with a peak RAM usage of 41.7 KB and flash memory usage of 460.1 KB, as compiled using the TensorFlow Lite Compiler.

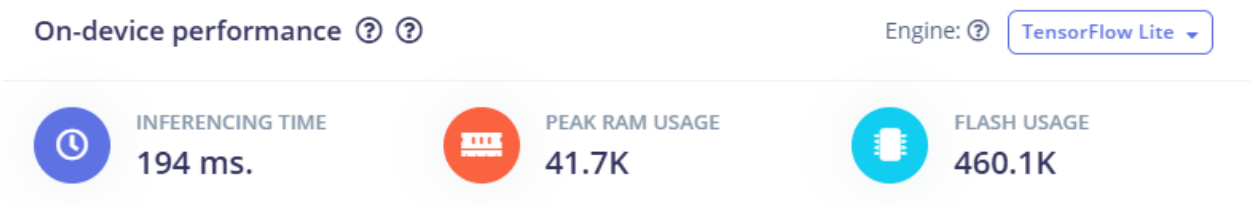

Figure 26: On-device performance metrics (inference time, RAM, flash usage)

5.3.8 Summary of Iteration 2

Iteration 2 focused on the development of a custom lightweight convolutional neural network, SimpleCNN, designed specifically for handwritten letter recognition with embedded deployment in mind. The architecture, implemented using PyTorch, was designed to maintain high accuracy while minimizing complexity. The EMNIST dataset was used for training, and a preprocessing pipeline was applied to ensure data compatibility and generalization.

The prototype system was first validated on a GUI and then successfully deployed onto an ESP32-CAM using TensorFlow Lite for Microcontrollers (TFLM). The model achieved a final accuracy of 92.80% after 25 epochs, with loss values steadily decreasing, confirming model convergence.

Despite strong performance, some letters—particularly L, I, G, and Q—showed lower classification accuracy. Confusion matrix analysis identified overlap between visually similar characters, revealing specific weaknesses in recall and precision for those classes.

Finally, on-device evaluation confirmed the model's suitability for embedded systems, with an inference time of 194 ms, peak RAM usage of 41.7 KB, and flash usage of 443.1 KB.

6. Discussion

6.1 Comparison of Iteration 1 & Iteration 2

Table 4: Comparison of metrics between 2 architectures

Feature	Iteration 1: MobileNetV2	Iteration 2: SimpleCNN
Architecture	Pre trained MobileNetV2	Custom lightweight CNN
Peak RAM Usage	86.2KB	41.7KB
Input Image Size	28x28 grayscale	28x28 grayscale
Number of Classes	26 (A-Z)	26 (A-Z)
Accuracy	87.4%	92.8%
Inference Time	~290ms	~194ms
Deployment Tool	Edge Impulse	".h" file (Arduino)

The comparison presented in Table 4 highlights the practical trade-offs between the two convolutional neural network architectures explored in this project: MobileNetV2 and SimpleCNN. While both models share the same input resolution (28×28 grayscale) and classification scope (26 uppercase letters), they differ significantly in terms of resource usage, accuracy, and deployment approach.

MobileNetV2, a pre-trained and widely adopted lightweight architecture, was integrated using Edge Impulse, simplifying the deployment process. However, its memory footprint is relatively large, with a peak RAM usage of approximately 86.2 KB. Its classification accuracy reached 87.4%, demonstrating solid performance given the hardware limitations. Nevertheless, the larger model size and external tool dependency limited flexibility in customizing the deployment pipeline.

In contrast, the SimpleCNN architecture was explicitly designed for the ESP32-CAM's constraints. With a significantly lower RAM requirement (41.7 KB), it provides a more memory-efficient alternative while achieving higher accuracy (92.8%) in simulated tests. The custom implementation using PyTorch to C translation allowed finer control over the model structure and quantization, but required more development effort.

Inference time for both models met real-time requirements. Overall, SimpleCNN demonstrated a better balance between efficiency and accuracy for embedded deployment, while MobileNetV2 offered easier integration but slightly lower performance within the same constraints.

6.2 Related work

Various efforts in recent years on this specific subject of deploying machine learning models for real-time handwriting recognition on embedded platforms have been made. Two studies are particularly close and relevant to this thesis.

The first, by I. Mchichou et al [22], implemented a CNN-based digit recognition system on a Raspberry Pi 3, achieving 99.94% accuracy with a custom architecture trained on the MNIST dataset. D. Nunez and S. Hosseini [23] developed a similar system for real-time letter recognition on the same platform, reaching 93.4% accuracy using a custom lightweight CNN model trained on the EMNIST dataset.

While both approaches demonstrated high recognition performance, they were built on the Raspberry Pi 3 platform. In contrast, this thesis explores a solution based on the ESP32-CAM, a more constrained and affordable microcontroller.

Table 5: Specifications comparison between ESP32-CAM & Raspberry Pi 3

	ESP32-CAM	**Raspberry Pi 3**
CPU	Dual-core	Quad-core
Clock Speed	160 MHz - 240 MHz	1.2 GHz
RAM	520 KB	1 GB
Flash	4 MB	8GB or more

Looking at Table 5, it shows that the ESP32-CAM operates with a dual-core processor at up to 240 MHz and only 520 KB of RAM, whereas the Raspberry Pi 3 provides a 1.2GHz quad-core CPU and 1 GB of RAM. Furthermore, the flash memory of the ESP32-CAM typically comes with 4MB, whereas for the Raspberry Pi 3 it is at least 8 GB. The ESP32-CAM integrates a camera, wireless connectivity, and onboard flash storage into a single low-cost package. The Raspberry Pi 3, in comparison, requires separate purchases to achieve similar functionality, including a camera module and microSD card. With these higher specifications, the Raspberry Pi 3 is more capable of executing this task. However, it also makes the Raspberry Pi 3 more expensive, with the cost of a complete setup potentially reaching ten times higher than the ESP32-CAM.

This study, therefore, distinguishes itself by implementing handwritten letter recognition on a platform that is not only more resource-limited but also far more economical, offering insights into how embedded machine learning can function under tighter hardware constraints.

6.3 Limitations

While the system successfully demonstrated the feasibility of deploying a convolutional neural network on the ESP32-CAM for handwritten letter recognition, certain limitations were observed during model development. The training process was constrained by the platform used, Edge Impulse, which imposes a maximum training duration of 20 minutes. This restriction limited the complexity of model architectures and the size of the dataset that could be effectively trained. As a result, the final model was trained on a reduced dataset containing only uppercase letters. Although this approach ensured balanced class representation, the reduced diversity may still influence the model's ability to generalize to varied handwriting styles.

Although the overall F1 score was relatively high, certain classes with visually similar shapes did experience misclassification, as evident in the confusion matrix and training results. Image capture was sensitive to environmental factors. Variations in lighting, shadows, and background noise affected classification accuracy.

6.4 Decisions on Model Optimizations

The design goal of this project was not to develop a highly accurate or complex model, but rather to explore how a simple, functional system could be realized under practical constraints. The objective was to create a lightweight character recognition model that performs reliably on the ESP32-CAM while keeping the development pipeline efficient and accessible.

A key design decision involved simplifying the dataset. The original dataset includes 62 classes comprising uppercase letters, lowercase letters, and digits. Initial experiments using the full dataset resulted in moderate accuracy and limited training progress. More importantly, the training environment provided by Edge Impulse imposes a 20-minute time limit per training session, restricting opportunities for tuning and experimentation. To address these constraints, the dataset was reduced to only uppercase letters, resulting in 26 classes. This made it possible to increase the number of training epochs and improve model stability while lowering the complexity of the classification task.

Model selection also reflected this streamlined approach. MobileNetV2 and a custom-designed SimpleCNN were chosen because they are lightweight and compatible with embedded deployment.

Larger architectures such as ResNet or VGG were not considered due to their incompatibility with the ESP32-CAM's resource limitations. Quantization into int8 format further reduced memory usage and allowed real-time inference, although this may have slightly impacted accuracy for certain visually similar letters.

Overall, the focus was on building a system that achieves consistent and reliable performance within practical boundaries. By emphasizing feasibility and responsiveness, the project demonstrates that effective embedded handwriting recognition is achievable without requiring high-end hardware or cloud resources.

6.5 Threats to Validity

One potential threat to validity arises from the reduction of the original EMNIST dataset to a smaller subset consisting of only uppercase letters. While EMNIST itself is a comprehensive and diverse dataset, this manual narrowing process may have inadvertently reduced the variety of handwriting styles included in the training data. Consequently, the final dataset may not fully represent the diversity of letter formation patterns typically found in real-world handwriting, potentially limiting the model's generalizability to unseen or atypical inputs.

Another threat arises from the reliance on a relatively clean dataset (EMNIST). In real-world scenarios, handwriting may include noise, blur, or irregular writing instruments, which were not fully captured in the test conditions.

6.6 Robustness and practical factors

Although the system performs well under controlled conditions, several external factors influence its reliability in real-world scenarios. One significant limitation is the sensitivity of the ESP32-CAM's built-in camera to lighting conditions. Poor lighting or strong shadows can degrade image quality, leading to reduced classification accuracy. The fixed focus and limited dynamic range of the camera further constrain its ability to consistently capture clear and well-lit images, especially in varying environments.

In addition to lighting, handwriting quality plays a major role in recognition performance. Variations in stroke thickness, letter spacing, slant, and overall legibility can impact the model's ability to correctly interpret characters. While the EMNIST dataset includes diverse handwriting samples, it may not fully capture the range of handwriting styles encountered in practical use. As a result, accuracy may decrease when the system is exposed to writing styles that deviate significantly from those seen during training.

These factors underscore the importance of evaluating embedded handwriting recognition systems under diverse environmental conditions. Future iterations of the system could benefit from improved preprocessing techniques, adaptive thresholding based on ambient light, or user feedback loops to enhance robustness across different use cases.

6.7 Lessons learned

Simpler model architectures tend to generalize better on devices with limited computational resources. While deeper models can theoretically achieve higher accuracy, their complexity often leads to overfitting when trained on small datasets and exceeds the memory or processing limitations of devices like the ESP32-CAM. Choosing a lightweight architecture such as MobileNetV2 enabled a practical compromise between inference speed and accuracy.

The quality and focus of the dataset can outweigh its size. Early training sessions were conducted using a larger dataset that included both uppercase and lowercase letters, as well as digits, which led to suboptimal performance due to memory constraints and limited training time.

Interpreting model behavior through plots like confusion matrices proved critical for understanding it. Detailed per-class evaluations made it possible to pinpoint which letters were frequently misclassified and whether errors stemmed from false positives or false negatives. This level of interpretability was important for diagnosing misclassifications and adjusting the dataset or training parameters accordingly.

7. Conclusion

7.1 Answers to Research Questions

RQ1: What approaches can be used to train a machine learning model that recognizes handwritten letters while remaining efficient enough for microcontroller deployment?

A machine learning model can be effectively trained for handwritten letter recognition on microcontrollers by adopting a two-stage approach: (1) training on a high-resource system and (2) optimizing the deployment model. In this thesis, both a pre-trained MobileNetV2 and a custom SimpleCNN were used. The training process involved using labeled grayscale images from the EMNIST Letters dataset and applying supervised learning with cross-entropy loss.
To maintain high accuracy, training included:

- Data augmentation (e.g., rotation, flipping, noise injection) to increase robustness.
- Careful hyperparameter tuning (e.g., learning rate, number of epochs).
- Training for an adequate number of epochs (up to 25) to ensure convergence.

To achieve efficiency suitable for microcontroller deployment:

- The trained model was quantized to 8-bit integers.
- A smaller, simpler CNN architecture (SimpleCNN) was developed to reduce memory and computation requirements.

This combination of robust training and lightweight architecture enabled the model to retain high accuracy (up to 92.8%) while being deployable on a resource-constrained ESP32-CAM.

RQ2: What dataset types and preprocessing techniques are most effective in managing the wide variation in handwriting styles?

The EMNIST dataset, resized to 28x28 and normalized, provided strong handwriting diversity even after reduction (Section 5.2.2). Augmentation was key to improving generalization on embedded models. These techniques allowed the model to handle a wide range of input conditions more effectively, ensuring consistent classification accuracy even under varied lighting and writing styles.

RQ3: How can the model be optimized to run on a microcontroller with limited resources, such as memory, processing power, and camera?

Optimizing the model for a microcontroller involved several key strategies:

1. **Model Quantization:** Converting the model's weights from 32-bit floats to 8-bit integers significantly reduced memory usage and accelerated inference on the ESP32-CAM without severely impacting accuracy.

2. **Model Simplification:** A custom SimpleCNN was designed with only two convolutional layers and two fully connected layers, reducing both the number of parameters and computation time.
3. **Dataset Reduction:** Training on a reduced EMNIST subset (only uppercase letters) allowed for faster training and a smaller final model suitable for the ESP32-CAM's flash memory.
4. **Hardware-Aware Design:** On-device preprocessing (grayscale conversion and resizing) was implemented efficiently in C++ to offload processing from the host and streamline real-time prediction.
5. **Memory and Inference Profiling:** Tools like Edge Impulse and Arduino serial monitoring were used to measure RAM usage and flash usage, ensuring compliance with ESP32-CAM hardware constraints.

These optimizations collectively allowed the system to perform real-time letter recognition directly on-device, proving the feasibility of embedded AI applications on low-power hardware.

7.2 Contribution

This thesis contributes to the field of embedded machine learning by demonstrating the feasibility of deploying convolutional neural networks for handwritten letter recognition directly on low-power microcontrollers. It presents a complete methodology for quantizing, optimizing, and deploying CNNs within the strict memory and processing limitations of platforms like the ESP32-CAM, without reliance on external computation. The work serves as a practical reference for future applications of real-time embedded AI in resource-constrained environments.

7.3 Future work

Future work could explore expanding the model to support full alphanumeric input, including lowercase letters and digits, while maintaining real-time inference performance. One area worth addressing is ensuring that all preprocessing, such as resizing and grayscale conversion, remains fully implemented on the ESP32-CAM. This could be verified through logging or visual debugging directly on the device.

Another focus could be identifying and resolving the primary bottlenecks in real-time operation. Preliminary profiling suggests that inference time is the most time-consuming step, but further testing could reveal whether memory access or preprocessing contributes significantly to latency.

Although the EMNIST dataset used in this study was balanced for uppercase letters, future work should analyze class distribution in extended datasets to prevent potential biases. Additionally, investigating how lighting variations affect classification accuracy would improve system robustness in uncontrolled environments.

Beyond character recognition, the solution could be adapted for other embedded tasks such as license plate reading, classroom tools for early literacy, or assistive writing devices.

Finally, the evaluation could be broadened by exploring alternative performance metrics, including standard deviation-based measures of prediction confidence or prediction stability across repeated inputs. These additions would provide a more nuanced understanding of model behavior under realistic conditions.

8. References

[1]. C. Wu, et al, September 2014, “Handwritten Character Recognition by Alternately Trained Relaxation Convolutional Neural Network”. Available: https://ieeexplore.ieee.org/document/6981035

[2] S. Fauziah, August 2023, “Analysis of CNN Optimizer for Classifying Letter Images”.Available: https://ieeexplore.ieee.org/document/10277347

[3] N. Alqahtani et al, January 2023, “Detection of Dyslexia Through Images of Handwriting using Hybrid AI Approach”. Available: https://www.researchgate.net/publication/375210757_Detection_of_Dyslexia_Through_Images_of_Handwriting_using_Hybrid_AI_Approach

[4] S. Vijayasharmila, et al, March 2023, “OPTICAL CHARACTER RECOGNITION FOR DYSLEXIA”. Available: https://www.irjmets.com/uploadedfiles/paper//issue_3_march_2023/35198/final/fin_irjmets1680763547.pdf

[5] S. Tabassum et al, March 2022, “An online cursive handwritten medical words recognition system for busy doctors in developing countries for ensuring efficient healthcare service delivery”. Available: https://www.nature.com/articles/s41598-022-07571-z

[6] V. Agrawal, June 2024, “Exploration of advancements in handwritten document recognition techniques”. Available: https://www.sciencedirect.com/science/article/pii/S2667305324000346

[7] M. Shafiq, S. Khan, and M. Ahmad, “Microcontroller,” in Microprocessors and Microcontrollers, vol. 1144, S. S. Rautaray, H. Pandey, and S. Das, Eds. Cham: Springer, 2024, pp. 107–126. Available: https://doi.org/10.1007/978-3-031-79582-4_7

[8] M. Nadeski, February 2020. Bringing machine learning to embedded systems. Available: https://docs.ampnuts.ru/ti.com.datasheet/AM5729/White_paper_SWAY020A.PDF

[9] E. Batzolis et al, March 2023, “Machine Learning in Embedded Systems: Limitations, Solutions and Future Challenges”. Available: https://ieeexplore.ieee.org/document/10099348

[10] A. Vasilyev, November 2021, “CNN optimizations for embedded systems and FFT”. Available: http://vision.stanford.edu/teaching/cs231n/reports/2015/pdfs/tema8_final.pdf

[11] Fetzer, J.H, 1990. “What is Artificial Intelligence?”. In: Artificial Intelligence: Its Scope and Limits. Studies in Cognitive Systems, vol 4. Springer, Dordrecht.
Available: https://doi.org/10.1007/978-94-009-1900-6_1

[12] R. Khalkar, A. Dikhit, June 2021, “Handwritten Text Recognition using Deep Learning.(CNN & RNN)”.
Available:https://www.researchgate.net/figure/enn-Diagram-for-AI-ML-NLP-DL_fig1_353939315

[13] I. El Naqa & M. J. Murphy, “What Is Machine Learning?,” in Machine Learning in Radiation Oncology: Theory and Applications, I. El Naqa, R. Li, and M. J. Murphy, Eds., Cham: Springer, 2015, pp. 3–11.
Available: https://link.springer.com/chapter/10.1007/978-3-319-18305-3_1

[14] T. Amaratunga, “What Is Deep Learning?,” in Deep Learning on Windows, Cham: Apress/Springer, 2021, pp. 1–14.
Available: https://link.springer.com/chapter/10.1007/978-1-4842-6431-7_1

[15] J. Wu, May 2017, “Introduction to Convolutional Neural Networks”.
Available: https://project.inria.fr/quidiasante/files/2021/06/CNN.pdf

[16] A. Roa, et al, 2014, “Automatic detection of invasive ductal carcinoma in whole slide images with Convolutional Neural Networks”. Available:
https://www.researchgate.net/figure/layer-CNN-architecture-composed-by-two-layers-of-convolutional-and-pooling-layers-a_fig3_263052166

[17] J. Stankovic, March 1996. “Real-Time and Embedded Systems”.
Available: https://dl.acm.org/doi/pdf/10.1145/234313.234400

[18] Al-Mahmud, A. Tanvin and S. Rahman, "Handwritten English Character and Digit Recognition," 2021.
Available: https://ieeexplore.ieee.org/document/9641160

[19] D. Fernandez, S. Hosseini, 2018, “Real-Time Handwritten Letters Recognition on an Embedded Computer Using ConvNets ”.
Available: https://ieeexplore.ieee.org/document/8592981

[20] I. Guyon & A. Elisseeff, “An Introduction to Feature Extraction,” in Feature Extraction: Foundations and Applications, M. Nikravesh, S. Gunn, L. A. Zadeh, and I. Guyon, Eds., vol. 207, Berlin, Heidelberg: Springer, 2006, pp. 1–25.
Available: https://link.springer.com/chapter/10.1007/978-3-540-35488-8_1

[21] I.Belcic, 2024 “What is classification in machine learning?”.
Available: https://www.ibm.com/think/topics/classification-machine-learning

[22] J.Murel, May 2024. “What is data augmentation?”.
Available: https://www.ibm.com/think/topics/data-augmentation

[23] J.Murel, January 2024. “What is a confusion matrix?”.
Available: https://www.ibm.com/think/topics/confusion-matrix

[24] L. Wulfert, C. Wiede, M. H. Verbunt, P. Gembaczka and A. Grabmaier. 2022. "Human Detection with A Feedforward Neural Network for Small Microcontrollers". Available: https://ieeexplore-ieee-org.proxy.mau.se/document/9924667

[25] T. V. A. Tong, B. Bui-Thanh and P. N. T. H. 2024. "Human Activity Recognition using Wireless Signals and Low-Cost Embedded Devices".
Available: https://ieeexplore-ieee-org.proxy.mau.se/document/10634669

[26] I. Mchichou, M. Tahiri, S. Amakdouf, et al, 2024, “Real-time Handwritten Digit Recognition Using CNN on Embedded Systems ”. Available: https://ieeexplore.ieee.org/document/10620086

[27] Al-Mahmud, Asnuva Tanvin, Sazia Rahman, 2021, “Handwritten English Character and Digit Recognition ”. Available: https://ieeexplore.ieee.org/document/10620086

[28] P. Latchoumy, G. Kavitha, S. Anupriya, H. Shaila Banu, 2022, “Handwriting Recognition using Convolutional Neural Network and Support Vector Machine Algorithms”.
Available: https://ieeexplore.ieee.org/document/9641160

[29] P. Janssen, 2014. “A design method and computational architecture for generating and evolving building designs”. Available: https://www.researchgate.net/figure/layer-CNN-architecture-composed-by-two-layers-of-convolutional-and-pooling-layers-a_fig3_263052166

[30] J. F. Nunamaker, M. Chen, and T. D. M. Purdin, “Systems development in information systems research,” Journal of Management Information Systems, vol. 7, no. 3, pp. 89–106, 1990. Available: http://www.jstor.org/stable/40397957

[31] B. Linders, January 2024, “Using Machine Learning on Microcontrollers: Decreasing Memory and CPU Usage to Save Power and Cost”. Available: https://ieeexplore.ieee.org/abstract/document/9912325

[32] S. Saha, et al, October 2022, “Machine Learning for Microcontroller-Class Hardware: A Review”. Available: https://ieeexplore.ieee.org/document/8765944

[33] K. Dokic, June 2020, “Microcontrollers on the Edge – Is ESP32 with Camera Ready for Machine Learning?”. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC7340922/

[34] X.Xie, et al, October 2022, “MobileNetV2 Accelerator for Power and Speed Balanced Embedded Applications”. Available: https://ieeexplore.ieee.org/document/9988258

[35] Yong Du, Minghua Wu, March 2021, “Deep Learning Classification of Systemic Sclerosis Skin Using the MobileNetV2 Model”. Available: https://www.researchgate.net/figure/The-proposed-MobileNetV2-network-architecture_fig1_350152088

[36] C. Crawfood, 2018, “EMNIST (Extended MNIST)”. Available: https://www.kaggle.com/datasets/crawford/emnist