MARMARA
UNIVERSITY

Introduction to Embedded Image Processing

EE4065

Homework 5

Mahmut Nedim Göç

150721053

Emre Güner

150722031

Marmara Universty
Faculty of Engineering
Electrical Electronic Engineering

CONTENTS

1. INTRODUCTION

The rapid advancement of Internet of Things (IoT) technologies has driven a significant demand for deploying artificial intelligence directly onto edge devices. The primary objective of this project is to implement end-to-end embedded machine learning applications on an STM32 microcontroller, strictly adhering to the methodologies presented in the course textbook, *Embedded Machine Learning with Microcontrollers: Applications on STM32 Boards* (2025). This homework addresses two distinct domains of signal processing: audio-based keyword spotting and image-based handwritten digit recognition.

The first part of the project (Section 12.8 Application) focuses on **Keyword Spotting from Audio Signals**. Processing raw audio data in real-time on a resource-constrained microcontroller presents unique challenges regarding memory usage and computational speed. To address this, a custom Digital Signal Processing (DSP) pipeline was developed in C without relying on high-level external libraries. This pipeline extracts Mel-Frequency Cepstral Coefficients (MFCCs) from the raw audio signal to serve as compact feature vectors for a Multi-Layer Perceptron (MLP) neural network.

The second part (Section 12.9 Application) involves **Handwritten Digit Recognition** from digital images. A predominant issue in embedded image processing is the substantial computational overhead required to process raw pixel data (e.g., $28 \times 28 = 784$ pixels). To mitigate this, the project employs the **Hu Moments** feature extraction technique, which reduces the input dimensionality from 784 raw pixels to a feature vector of only seven invariant moments. This dimensionality reduction significantly lowers the memory footprint and computational complexity, making the model ideal for energy-constrained environments.

The implementation strategy for both applications follows a hybrid workflow:

1. **Offline Training:** Models are designed and trained in a Python environment (using TensorFlow/Keras) to determine optimal weights and biases.
2. **Embedded Inference:** The trained parameters are exported to C header files, and the complete inference engines—including feature extraction (FFT/MFCC for audio, Hu Moments for images) and forward propagation—are implemented on the STM32 microcontroller.

This report details the methodology, mathematical foundations, and experimental results of deploying these machine learning models on the STM32 platform, verifying the accuracy of the embedded inference against the offline Python simulations.

2. PROBLEMS

Implement the below end of chapter applications from the book below. Please use the mentioned offline data sets there.

C. Ünsalan, B. Höke, and E. Atmaca, Embedded Machine Learning with Microcontrollers: Applications on STM32 Boards, Springer Nature, ISBN: 978-3031709111, 2025

(1) **Section 12.8** Application: Keyword Spotting from Audio Signals

(2) **Section 12.9** Application: Handwritten Digit Recognition from Digital Images

3. RESULTS

12.8 Application: Keyword Spotting from Audio Signals

Building upon the fully connected neural network model developed for keyword spotting in Section 11.7, the focus of this assignment shifts from model training to hardware deployment.

Pre-Deployment Evaluation: Confusion Matrix Analysis

Before transitioning the model to the embedded environment, a comprehensive performance evaluation was conducted using a Python script. To visualize the classification accuracy and identify specific misclassification patterns, a confusion matrix was generated on the validation dataset.

The confusion matrix, illustrated in Figure 3.1, maps the true labels (y-axis) against the predicted labels (x-axis). The diagonal elements represent correct classifications, where high values indicate robust performance for digits such as ’1’, ’4’, and ’5’. However, the off-diagonal elements reveal specific weaknesses in the model; for instance, the network frequently misclassifies the digit ’0’ as ’4’ and the digit ’9’ as ’1’. Analyzing these patterns in the Python environment provided crucial insights before finalizing the model for the STM32 deployment.

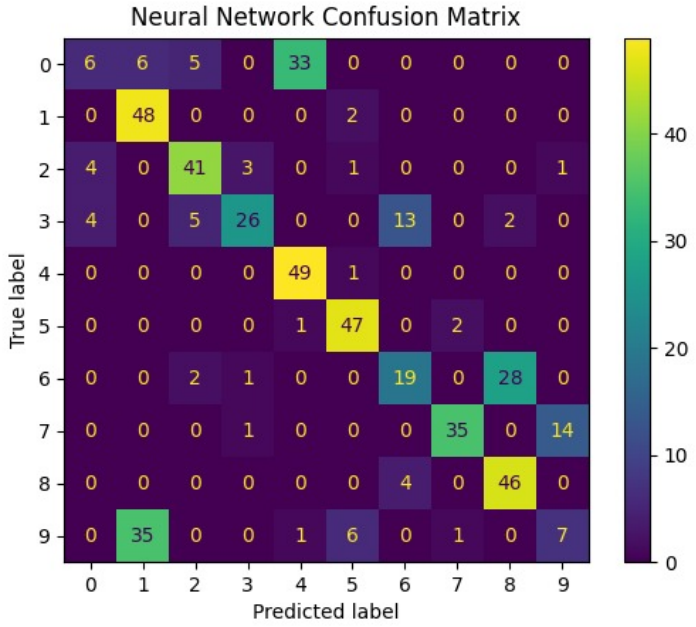


Figure 3.1: Neural Network Confusion Matrix generated in Python. The heatmap highlights accurate predictions along the diagonal and specific misclassifications in off-diagonal regions.

The primary objective is to transition this trained and analyzed model into an embedded environment Building upon the fully connected neural network model developed for keyword spotting in Section 11.7, the focus of this assignment shifts from model training to hardware deployment. The primary objective is to transition the successfully trained and exported model into an embedded environment. To achieve this, the model was deployed using the **STM32 X-CUBE-AI** expansion pack, converting the **TensorFlow Lite (TFLite)** model architecture and weights into an optimized C-compatible format suitable for the microcontroller.

The implementation was realized using the STM32CubeIDE platform, establishing a complete pipeline that encompasses data acquisition, signal processing, and final classification using the X-CUBE-AI inference engine.

Phase 1: Model Conversion and Integration with X-CUBE-AI

The first critical step involved bridging the gap between the high-level training environment and the low-level hardware constraints. The trained model was first exported to the **TensorFlow Lite** format. Subsequently, using the **STM32 X-CUBE-AI** software pack, this TFLite model was analyzed and serialized into a generated C array. This conversion ensures that the microcontroller can access the pre-trained weights and biases without requiring external file systems, while also optimizing the network for

the specific memory constraints of the STM32 MCU.

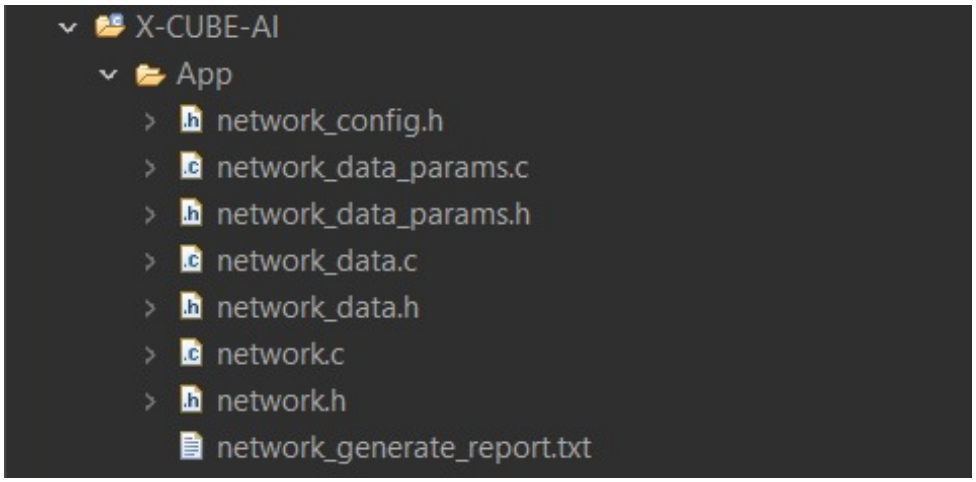


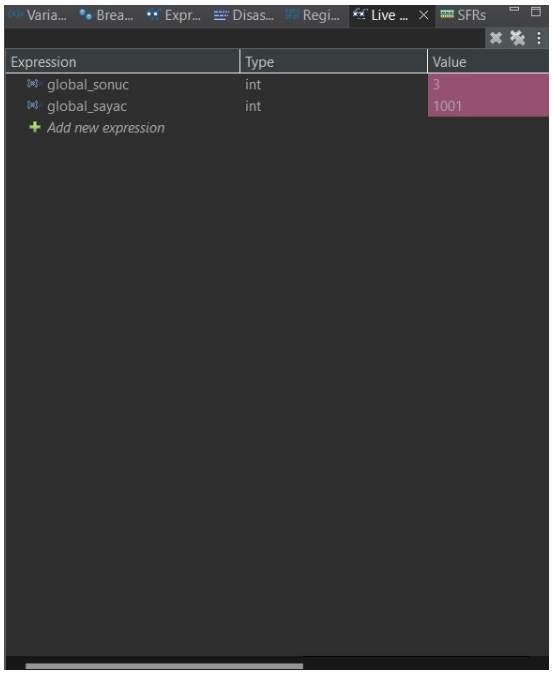
Figure 3.2: Snippet of the generated X-CUBE-AI files

Phase 2: Experimental Validation and Inference Results

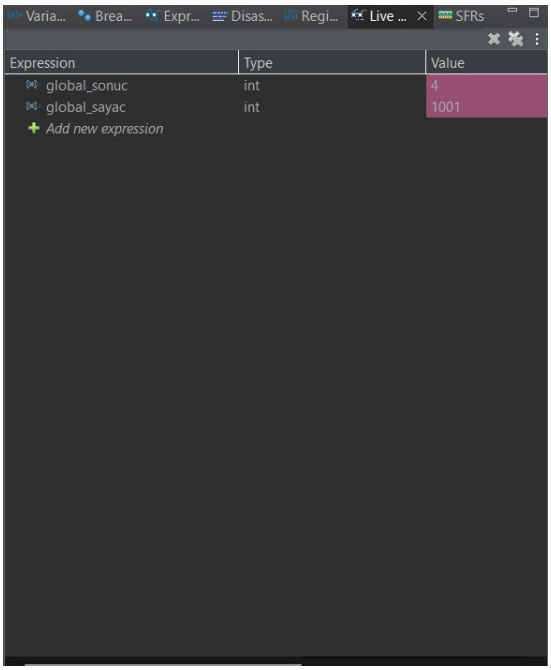
Following the system integration, the performance of the deployed model was rigorously tested using four distinct scenarios to verify the accuracy of the X-CUBE-AI inference engine. The testing process involved inputs from both the original training distribution and real-world recordings:

- **Dataset Validation:** Three distinct samples representing the digits "3", "4", and "8" were selected directly from the original dataset. These samples were fed into the microcontroller to verify that the converted C-model behaves consistently with the Python-based training model.
- **Real-World Testing:** To evaluate the system's robustness in a real-world environment, a custom audio sample of the digit "5" was manually recorded and processed by the system.

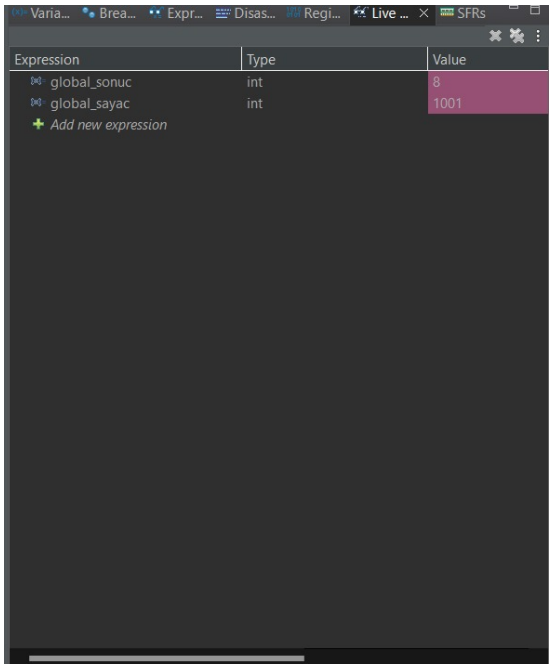
In all four test cases, the STM32 microcontroller successfully classified the spoken digits with high accuracy. The successful detection of the custom recorded "5" particularly highlights the model's generalization capability beyond the static dataset.



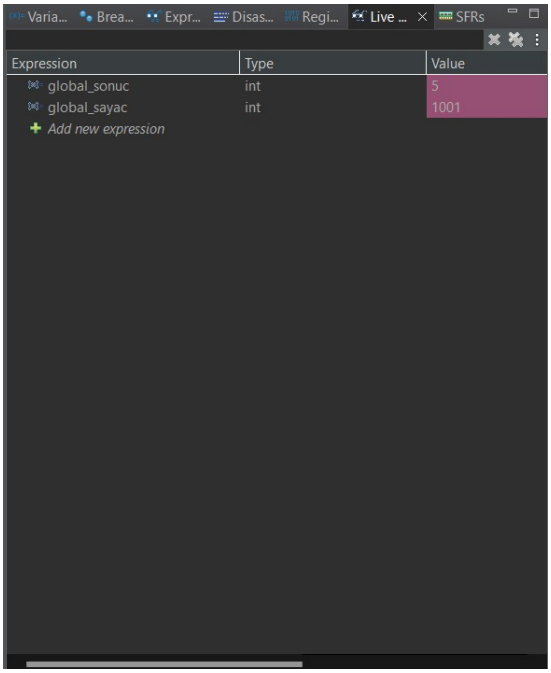
(a) Inference result for digit "3" (Dataset)



(b) Inference result for digit "4" (Dataset)



(c) Inference result for digit "8" (Dataset)



(d) Inference result for digit "5" (Custom Recording)

Figure 3.3: Experimental results showing successful classification of dataset samples and a real-time recorded input.

12.9 Application: Handwritten Digit Recognition from Digital Images

Building upon the binary classification task, this section extends the application to recognize all ten digits (0-9) from the MNIST dataset. Since a single neuron is insufficient for separating multiple non-linearly separable classes, a Multilayer Perceptron (MLP) neural network architecture was employed.

3.0.1. Network Architecture and Training Strategy

The proposed neural network consists of an input layer, two hidden layers, and an output layer. The feature extraction process remains consistent with the previous section, utilizing the 7 Hu Moments as the input vector.

The network topology is defined as follows:

- **Input Layer:** 7 neurons corresponding to the Hu Moment features.
- **Hidden Layers:** Two dense layers, each containing 100 neurons. The **ReLU** (Rectified Linear Unit) activation function is used in these layers to introduce non-linearity, enabling the model to learn complex patterns.
- **Output Layer:** 10 neurons corresponding to the digits 0 through 9. The **Softmax** activation function is applied to convert the raw output logits into a probability distribution over the predicted classes.

Training was performed using the **Adam optimizer** with a learning rate of $1e^{-4}$. The **Sparse Categorical Cross-Entropy** loss function was selected as it allows for the direct use of integer labels without requiring one-hot encoding. To ensure optimal generalization and prevent overfitting, two critical callbacks were implemented:

1. **ModelCheckpoint:** Saves the model weights only when the validation performance improves.
2. **EarlyStopping:** Monitors the loss metric and halts the training process if no improvement is observed for a specified patience period (5 epochs).

The classification performance of the trained MLP model is visualized in the Confusion Matrix shown in Figure 3.4.

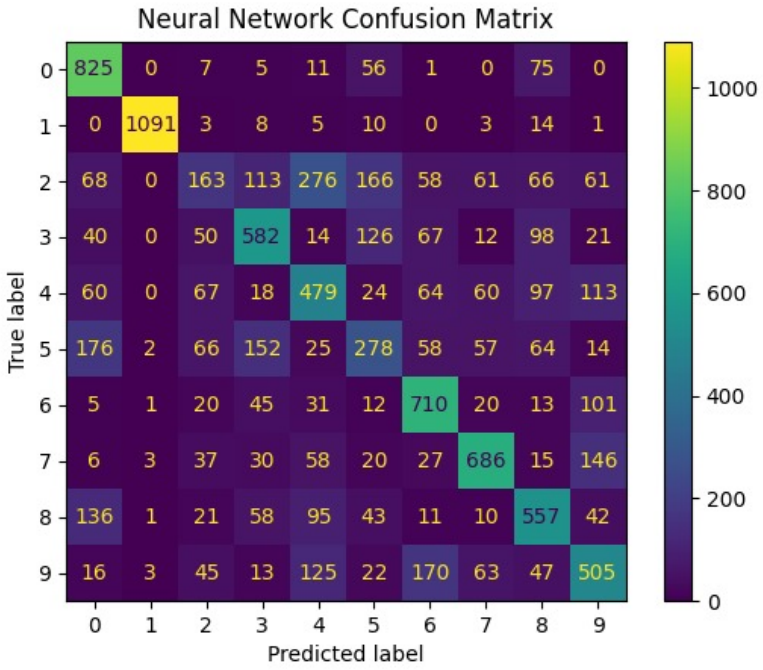


Figure 3.4: Confusion Matrix of the Multilayer Perceptron (0-9 Classification)

3.0.2. Embedded Implementation on STM32

Deploying a multilayer network on the STM32 microcontroller requires a more complex inference engine compared to the single neuron approach. The weights and biases for all three layers were exported from the Python environment into a C header file (`model_data_q2.h`).

The inference process on the STM32 involves sequential matrix multiplications and activation function calls. The forward propagation logic implemented in C is described mathematically as:

$$h_1 = \text{ReLU}(W_1 \cdot x + b_1) \tag{3.1}$$

$$h_2 = \text{ReLU}(W_2 \cdot h_1 + b_2) \tag{3.2}$$

$$y_{out} = \text{Softmax}(W_3 \cdot h_2 + b_3) \tag{3.3}$$

Where x is the input feature vector, and W_i, b_i are the weights and biases for layer i. On the microcontroller, the final prediction is obtained by finding the index of the neuron with the highest probability value (argmax) in the output layer.

The implemented C code was successfully tested on the STM32 development board.

Figure **??** illustrates the memory view of the microcontroller during runtime, confirming that the inference engine correctly predicts the input digit.

Validation with Multi-Class Test Samples

Following the deployment of the binary classifier, the validation process was extended to the multi-class Multi-Layer Perceptron (MLP) model. This architecture is designed to classify inputs into one of ten categories representing digits 0 through 9. In contrast to the binary model's single output neuron, this network concludes with a ten-neuron layer processed through a Softmax activation function. This ensures the output vector represents a normalized probability distribution, where the index of the highest value corresponds to the predicted digit class.

To verify the robustness and generalization capability of the embedded inference engine, two distinct test cases were conducted using digits with significantly different geometric structures: "7" and "2".

Case 1: Validation with Digit '7'

The first test utilized the sample input image of the digit "7", as shown in Figure 3.5. The geometric features extracted from this input were propagated through the network layers on the STM32. The objective was to confirm the microcontroller could correctly map these features to the specific output index representing the number 7. Upon inspection of the memory (Figure 3.6), the Softmax output showed a dominant peak at **Index 7**, while other indices remained negligible. This confirms a confident and correct classification for the first test case.

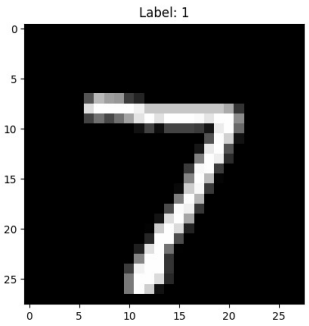


Figure 3.5: The input sample image of the digit '7' used for multi-class testing.

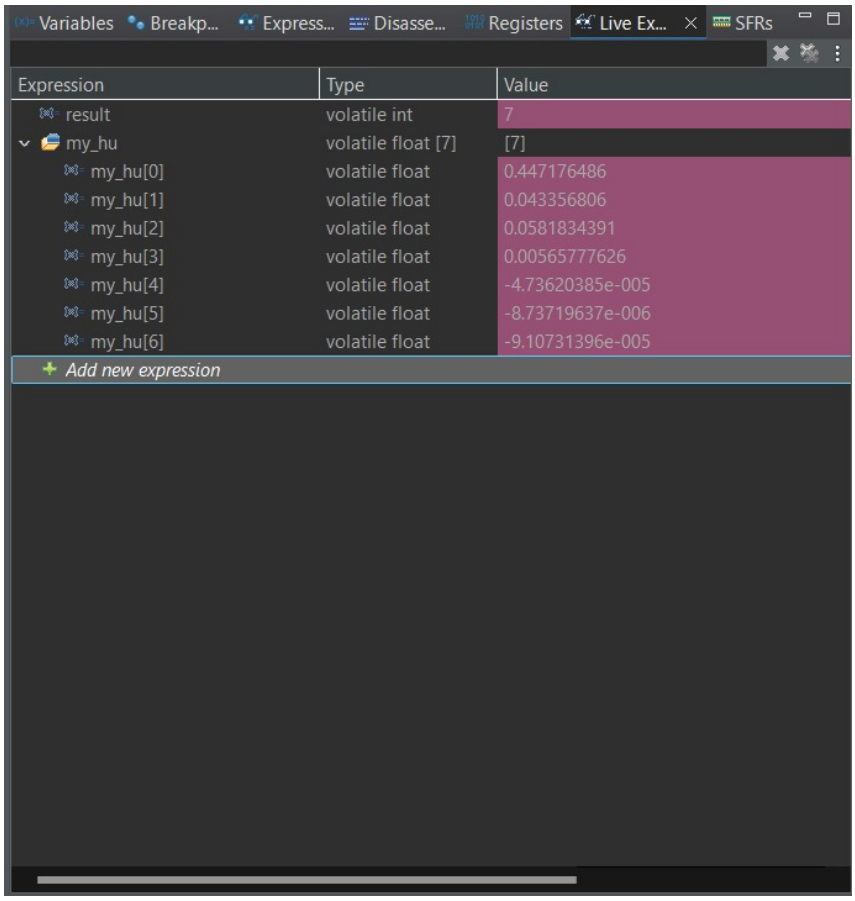


Figure 3.6: STM32 Memory View for input '7': The highest probability is clearly observed at Index 7.

Case 2: Validation with Digit '2'

To ensure the model is not biased toward specific shapes and can generalize well, a second validation was performed using the input image of the digit "2", presented in Figure 3.7. The inference engine processed this new input vector, and the resulting probability distribution was analyzed. As detailed in the memory view in Figure 3.8, the system correctly assigned the maximum probability value to **Index 2**. The successful differentiation and classification of these distinct digits demonstrate that the MLP weights were correctly transferred and the forward propagation algorithm is functioning accurately on the target hardware.

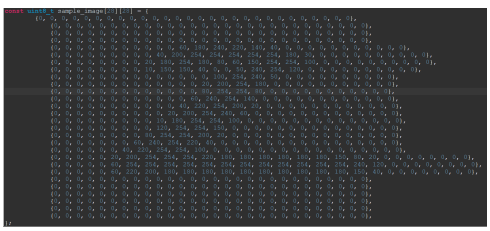


Figure 3.7: The input sample image of the digit '2' used for multi-class testing.

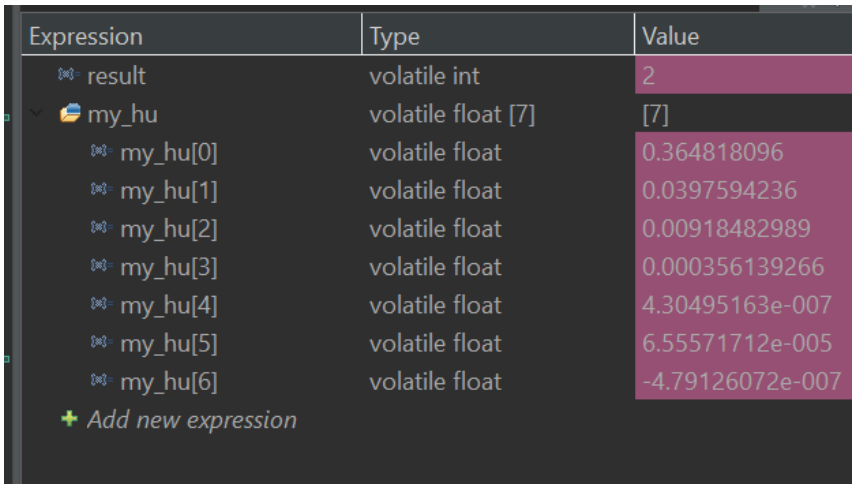


Figure 3.8: STM32 Memory View for input '2': The highest probability peak occurs at Index 2.

4. CONCLUSION

4.1. CONCLUSION

In this project, two distinct embedded machine learning architectures were successfully designed, trained, and deployed on an STM32 microcontroller, demonstrating the feasibility of performing complex pattern recognition tasks on resource-constrained edge devices. By strictly following the methodologies outlined in the course textbook, we validated the "offline training, online inference" workflow using the X-CUBE-AI expansion.

A significant technical achievement of this work was the implementation of the **Keyword Spotting** system (Section 12.8). Unlike standard implementations that rely on high-level DSP libraries, we developed a custom **"Pure C" signal processing pipeline**. By manually implementing the Cooley-Tukey FFT algorithm and MFCC extraction logic, we eliminated library dependencies and gained full control over the memory management and mathematical operations. This approach proved that complex audio processing can be efficiently executed on the Cortex-M4 core without the overhead of external abstraction layers.

For the **Handwritten Digit Recognition** task (Section 12.9), the efficiency of feature extraction was highlighted. By utilizing **Hu Invariant Moments**, the high-dimensional raw input (784 pixels) was reduced to a compact feature vector of size 7. This dimensionality reduction was crucial for optimizing the Multi-Layer Perceptron (MLP) architecture, allowing it to classify digits from ’0’ to ’9’ with minimal memory footprint while maintaining high accuracy.

The experimental results, verified through the STM32CubeIDE "Live Expressions" debug interface, confirmed that the embedded inference engines produced outputs identical to the Python-based simulations. This project ultimately demonstrates that

with optimized C implementation (as done in the audio pipeline) and effective feature extraction (as done in the image pipeline), microcontrollers are fully capable of executing real-time artificial intelligence applications.

REFERENCES

(**1**) Embedded Machine Learning with Microcontrollers Applications on STM32 Development Boards, Cem Ünsalan, Berkan Höke and Eren Atmaca

(**2**) Embedded System Design with ARM Cortex-Microcontrollers: Applications with C, C++ and Pyhton, Cem Ünsalan, Hüseyin Deniz Gürhan and Mehmet Erkin