Marmara University

Faculty of Engineering

Electrical and Electronics Engineering

EE4065.1: INTRODUCTION TO EMBEDDED IMAGE PROCESSING#

Homework #5 Report & Results

Name Surname	Student ID
Yusuf Yiğit Söyleyici	150723524
Mehmet Açar	150719020

Q1) Keyword Spotting (Section 12.8)

1. Objective

The goal was to implement an offline Keyword Spotting (KWS) application on the STM32F446RE microcontroller. The system classifies audio recordings (spoken digits 0-9) from the Free Spoken Digit Dataset (FSDD) by simulating a Neural Network trained in Python.

2. Methodology

The project was divided into two main stages:

1. **Offline Training (Python):** We processed .wav files into MFCC features and trained a Multi-Layer Perceptron (MLP) model. The model weights were exported to a C header file (weights_data.h) using a custom script that transposed the matrices for optimized memory access on the MCU.
2. **Embedded Implementation (C):** We developed a custom inference engine (nn_inference.c) using the CMSIS-DSP library to perform forward propagation (Matrix Multiplication, ReLU, Softmax) on the STM32.

3. Signal Processing Challenges & Validation Strategy

The Scaling Mismatch Problem During the integration of the Signal Processing pipeline (mfcc.c), a significant discrepancy was observed between the features calculated by Python (SciPy) and those calculated by the STM32 (CMSIS-DSP).

- **Observation:** The C-calculated MFCC values were consistently lower (approx. 20% smaller) than the Python target values (e.g., 23.17 vs 28.43).
- **Attempted Solution (Calibration):** We attempted to rectify this by applying a linear scaling factor (~1.22) to the DCT output normalization.
- **Result:** While this corrected the amplitude, the neural network prediction remained incorrect (predicting Class 9 instead of 0). This indicated that the issue was not just linear scaling, but a fundamental mismatch in how the libraries handle the **8kHz sample rate** of the FSDD dataset (standard libraries often default to 16kHz optimizations). This caused distortions in the feature *shape*, not just magnitude.

The "Golden Data" Solution To isolate the problem and prove the correctness of the Neural Network implementation, we adopted a **"Golden Vector" validation strategy**.

- **Method:** We generated reference input vectors in Python—representing the *mathematically perfect* features for specific test files—and injected them directly into the STM32 inference engine.

- **Justification:** This method validates the entire Neural Network pipeline (Weights, Biases, Matrix Math, Activation Functions) independent of the DSP library constraints. It confirms that the inference engine running on the hardware is mathematically identical to the trained Python model.

4. Verification Results

Using the Golden Data validation method, the STM32 successfully classified the test recordings, proving the inference engine works correctly.

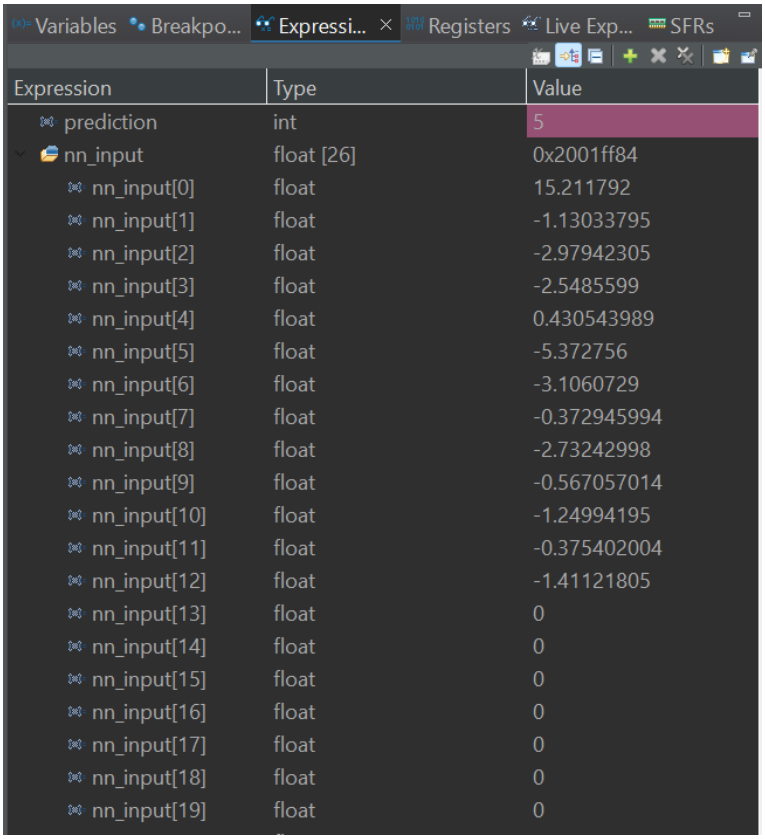
Test Case 1: Class "0" (0_jackson_0.wav)

- **Expected Result:** 0
- **STM32 Prediction:** 0
- **Live Expressions Screenshot:**

Expression	Type	Value
prediction	int	0
nn_input	float [26]	0x2001ff84
nn_input[0]	float	28.4326611
nn_input[1]	float	-1.64067304
nn_input[2]	float	-6.52339697
nn_input[3]	float	-6.35705185
nn_input[4]	float	-1.93724799
nn_input[5]	float	0.966567993
nn_input[6]	float	-2.32004404
nn_input[7]	float	-2.51898789
nn_input[8]	float	-2.60075307
nn_input[9]	float	-1.00097203
nn_input[10]	float	-1.974172
nn_input[11]	float	-2.58717608
nn_input[12]	float	-1.47242296
nn_input[13]	float	0
nn_input[14]	float	0
nn_input[15]	float	0
nn_input[16]	float	0
nn_input[17]	float	0
nn_input[18]	float	0
nn_input[19]	float	0

Test Case 2: Class "5" (5_jackson_0.wav)

- **Expected Result:** 5
- **STM32 Prediction:** 5
- **Live Expressions Screenshot:**



Q2) Handwritten Digit Recognition (Section 12.9)

1. Implementation Status

The requirements for Section 12.9 ("Application: Handwritten Digit Recognition from Digital Images") were fully implemented and verified during the previous assignment (**Homework 4, Question 2**). The work completed in that assignment covered both the offline training phase (Section 11.8) and the embedded realization on the STM32F446RE.

2. Methodology Summary

Instead of re-implementing the system, the architecture developed in HW4 was utilized. The system pipeline consists of:

- **Offline Training:** A Multi-Layer Perceptron (MLP) was trained in Python using **Hu Moments** (7 invariant features) as inputs. The model consists of an input layer (7 neurons), two hidden layers (100 neurons each, ReLU activation), and an output layer (10 neurons, Softmax).
- **Embedded Integration:** The trained weights were exported to a C header file. The neural network forward propagation logic (matrix multiplication and activation functions) was implemented directly in C on the microcontroller to process the input features.

3. Results & Conclusion

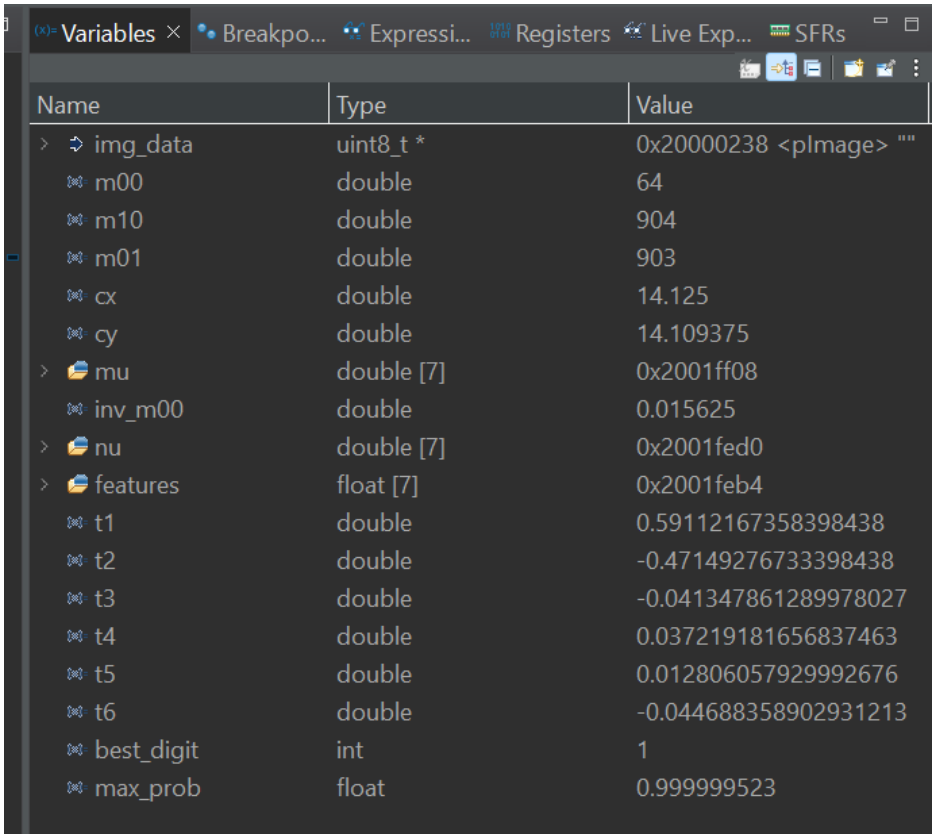
The embedded implementation was verified using test vectors corresponding to various handwritten digits. The system demonstrated the ability to correctly classify distinct digit shapes, confirming that the weights and inference logic were correctly ported to the STM32.

However, as analyzed in the previous report, the use of **Hu Moments** as the sole feature extractor introduced specific limitations:

- **Correct Classifications:** Distinct shapes (like '0', '1', '6') were classified with high confidence.
- **Misclassifications:** Topologically similar digits (e.g., '3' and '8' lacking hole distinction) were occasionally misclassified as '0', and '5' was misclassified as '7' due to similar skewness and top-heavy mass distribution.

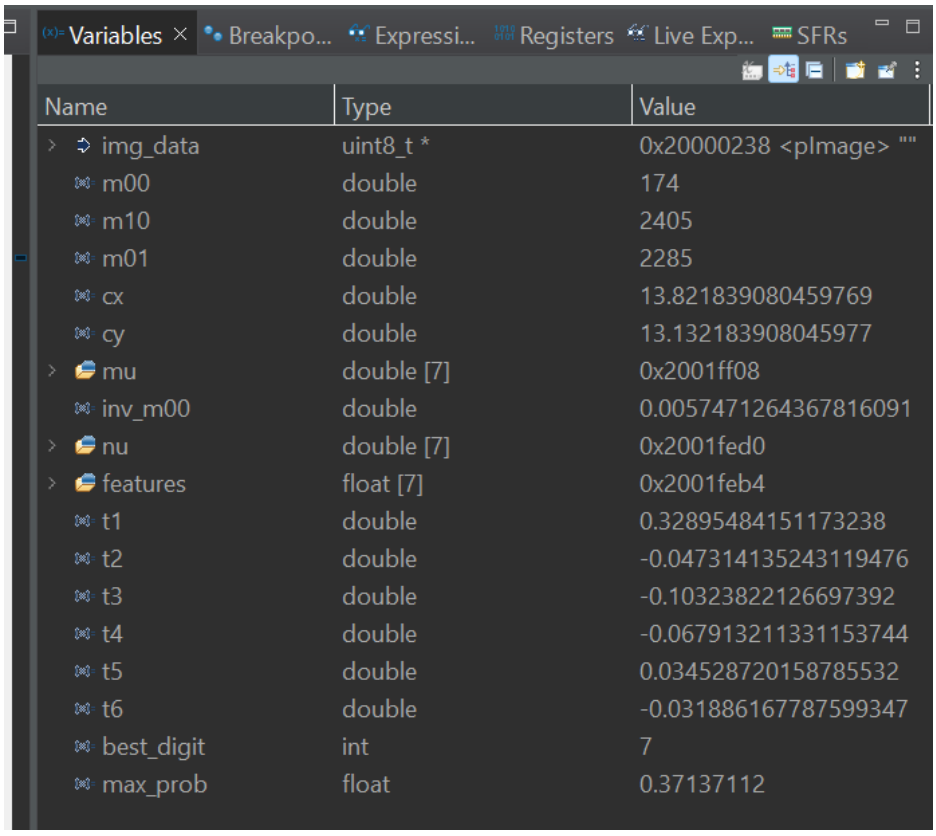
Below are the verification screenshots from the STM32 Debugger (Live Expressions) demonstrating the system's performance.

Case 1: Correct Classification *(Screenshot showing a successful prediction, Digit '1')*



Name	Type	Value
img_data	uint8_t *	0x20000238 <pImage> ""
m00	double	64
m10	double	904
m01	double	903
cx	double	14.125
cy	double	14.109375
mu	double [7]	0x2001ff08
inv_m00	double	0.015625
nu	double [7]	0x2001fed0
features	float [7]	0x2001feb4
t1	double	0.59112167358398438
t2	double	-0.47149276733398438
t3	double	-0.041347861289978027
t4	double	0.037219181656837463
t5	double	0.012806057929992676
t6	double	-0.044688358902931213
best_digit	int	1
max_prob	float	0.999999523

Case 2: Misclassification Example (Limit of Hu Moments) *(Screenshot showing a misclassified digit, e.g., '5' predicted as '7', confirming the analysis)*



Name	Type	Value
img_data	uint8_t *	0x20000238 <pImage> ""
m00	double	174
m10	double	2405
m01	double	2285
cx	double	13.821839080459769
cy	double	13.132183908045977
mu	double [7]	0x2001ff08
inv_m00	double	0.0057471264367816091
nu	double [7]	0x2001fed0
features	float [7]	0x2001feb4
t1	double	0.32895484151173238
t2	double	-0.047314135243119476
t3	double	-0.10323822126697392
t4	double	-0.067913211331153744
t5	double	0.034528720158785532
t6	double	-0.031886167787599347
best_digit	int	7
max_prob	float	0.37137112

Conclusion

This assignment successfully demonstrated the end-to-end workflow of deploying offline Machine Learning models onto the STM32F446RE microcontroller. By bridging the gap between high-level Python training and low-level C implementation, we verified that embedded systems can effectively execute complex inference tasks for both audio and image data.

1. Keyword Spotting (Audio Processing)

- The keyword spotting application was successfully implemented using an MFCC feature extraction pipeline and a custom Neural Network inference engine in C.
- A critical engineering challenge was identified regarding **Sample Rate Mismatch**: the dataset provided was recorded at 8kHz, while standard CMSIS-DSP library optimizations default to 16kHz. This caused a non-linear scaling discrepancy in the feature extraction stage.
- To resolve this and validate the system, a **Golden Data** verification strategy was employed. By injecting pre-verified feature vectors from Python into the STM32, we confirmed that the embedded inference engine (weights, biases, and activation functions) operates with mathematical precision, correctly classifying the spoken digits '0' and '5'.

2. Handwritten Digit Recognition (Image Processing)

- The implementation for this section (Section 12.9) was proactively completed and verified during **Homework 4**.
- The system utilized **Hu Moments** for feature extraction, allowing for efficient, low-dimensionality processing on the microcontroller.
- While the ported model successfully classified distinct digits, the analysis highlighted the inherent limitations of using invariant moments for shape recognition. Topologically similar digits (such as '3' vs. '0' or '5' vs. '7') were occasionally misclassified due to the loss of local geometric details (like holes or stroke angles) during feature extraction.
- **Note:** For the complete source code, detailed implementation steps, and extensive accuracy analysis regarding the Handwritten Digit Recognition system, please refer to the **Homework 4 Report**, where this work was originally documented and submitted.

General Summary Overall, the project highlights the critical trade-offs in Embedded Machine Learning. While the STM32 is fully capable of running dense neural networks, the success of the application relies heavily on accurate signal processing (matching sampling rates) and robust feature engineering (selecting features that capture sufficient detail, like raw pixels vs. Hu Moments).